Moxa NPort Real TTY Driver for Arm-based Platform Porting Guide

Moxa Technical Support Team
support@moxa.com

Contents

Copyright © 2021 Moxa Inc. Released on August 10, 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: +886-2-8919-1230
Fax: +886-2-8919-1231

1 Introduction

This document is intended for programmers who are porting the NPort Real TTY 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
- The Yocto Project documentation
- 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 Real TTY driver to other platforms.

The Real TTY 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 Real TTY 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 Porting to the Moxa UC-Series—Arm-based Computer

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 Real TTY Driver to understand the requirement.

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

```
# ./mxinst
```

Copyright © 2021 Moxa Inc.

2.2 Cross-compiler and the Real TTY 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 all of 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 cross-compiler toolchain and the kernel source package webpage under the product page.

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

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 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-linuc-4.4 /moxa/kernel
   # cd /moxa/kernel
   # make uc3100_defconfig
   ```
   ← Replace the UC 3100 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 Real TTY driver with a cross-compiler,” to cross-compile Moxa's driver for the UC-Series platforms.

Copyright © 2021 Moxa Inc.

The NPort Real TTY driver, which includes the driver module, service daemons, and tools, needs to be compiled. The files are listed as follows:

- npreal2.ko: Real TTY kernel extension
- npreal2d: Daemon of Real COM communication
- npreal2d_redund: Daemon of Redundant COM mode only for the NPort CN2500/CN2600 Series.
- mxloadsvr: Daemons reloading tool.
- mxaddsvr: Port-mapping tool.
- mxdelsvr: Port-unmapping tool.
- mxsetsec: Secure mode setting tool.
- mxcfmat: Internal-use only tool.
- mxmknod: Internal-use only tool.
- mxrmnod: Internal-use only tool.
- npreal2d.cf: Configuration template.

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 Real TTY driver with a cross-compiler."

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

2.3 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 Real TTY driver source directory to cross-compile the MOXA driver.

The steps are as follows:

```
# ./mxcc
Enter target device architecture (ARCH) [arm]:
Enter cross-compiler (CROSS_COMPILE) [arm-linux-gnueabihf-]:
Enter target device kernel source directory [/moxa/kernel/]:
If you wish to use secure communication with the NPort 6000 Series device,
choose [Y] to enable the SSL function.
Note: This function supports Real COM with secure mode in the NPort 6000
Series only.
Do you want to enable secure mode? [Y/N]: N
The polling mode allows you to open the tty port as nonblocking even if
```

Copyright © 2021 Moxa Inc.

```
the NPort is
not connected.
Do you want to set the driver to polling mode? [Y/N]: N
*****************************************************************
MOXA NPort Server Real TTY Driver Series driver cross-compiling finished.
When cross compiling is successful, the driver is outputted to output
folder.
*****************************************************************
```

The binaries will now be generated and placed in the output directory under the source code folder.

2.4 Manually build the Real TTY driver with a cross-compiler

2.4.1 To cross-compile npreal2 driver, users can find "Makefile" in the driver source folder, then run it.

```
# make -C KDIR=<KERNEL_SOURCE> M=<DRIVER_SOURCE> ARCH=<ARCH>
CROSS_COMPILE=<CROSS_COMPILE>  KVER_MAJOR=<KERNEL_MAJOR>
KVER_MINOR=<KERNEL_MINOR> modules
```

<KERNEL_SOURCE>: The directory of target kernel source.

<DRIVER_SOURCE>: The directory of the Real TTY driver source.

<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 the path of toolchain exists in your PATH environment variable, please enter "arm-linux-gnueabihf-" here.

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

For example:

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

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

Copyright © 2021 Moxa Inc.

For example:

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

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

```
# make -C KDIR=/moxa/kernel M=/home/user/moxa/source ARCH=arm
CROSS_COMPILE=arm-linux-gnueabihf- KVER_MAJOR=4 KVER_MINOR=4
modules
```

After using the "make" command to cross-compile the drivers, the driver file "npreal2.ko" can be found in the source code directory.

2.4.2 To cross-compile the daemons and tools, please find "Makefile" in the driver source folder, then run it.

```
# make <TARGET> CROSS_COMPILE=<CROSS_COMPILE> CC=<C_COMPILE>
CFLAGS=<C_FLAGS>
```

<TARGET>: Set one of npreal2d, preal2d_redund, and tools.

<CROSS_COMPILE>: The cross-compile toolchain path. If the toolchain is "arm-linux-gnueabihf", and the path of toolchain exists in your PATH environment variable, please enter "arm-linux-gnueabihf-" here.

<C_COMPILE>: The C compiler offered by the cross-compiler toolchain. It is "gcc" if the toolchain is "arm-linux-gnueabihf-".

<C_FLAGS>: Please specify the preprocessor definitions of Real TTY driver here.

Note: "-DNO_INIT" must be included or else the cross-compiler may return error messages.

Please see the definitions:

- "-DNO_INIT": Disable the startup service.
- "-DOFFLINE_POLLING": Allow tty not to be blocked if the NPort is offline.

e.g.: To build TARGET=npreal2d with a polling feature, please use the following command:

```
# make npreal2d CROSS_COMPILE="arm-linux-gnueabihf-" CC=gcc
CFLAGS="-DNO_INIT -DOFFLINE_POLLING"
```

After using the "make" command to cross compile the daemons and tools, the binaries can be found in the source code directory.

Copyright © 2021 Moxa Inc.

2.4.3 (Optional) Build a secure mode connection to the NPort 6000 Series

When it is required to use a secure mode connection to the NPort 6000 Series, the npreal2d daemon should be built manually because it needs extra OpenSSL library. This section introduces the secure mode npreal2d building in addition to the OpenSSL library demonstration. OpenSSL is maintained by www.openssl.org.

Most of the Linux distributions have package management tools, such as apt-get or yum, which help you to install OpenSSL library and development tools. In an Arm platform, it has to be built from the source code. You may refer to OpenSSL's user guide to generate the library first. The instructions may vary amongst different OpenSSL versions, cross-compilers, or building hosts.

The demonstration here illustrates the process that Moxa has built for the library for Real TTY driver and for the Moxa's lab testing.

1. Create the folders below for OpenSSL products:

```
$ cd ~
$ mkdir openssl-lib
$ cd openssl-lib
$ mkdir openssl-arm
$ mkdir ssl-arm
```

2. Check out the OpenSSL source code. We used a stable branch named OpenSSL-fips-2_0_9. The command below will download the OpenSSL-fips-2_0_9 source code in the openssl folder.

```
$ git clone https://github.com/openssl/openssl.git -b OpenSSL-fips-2_0_9
```

3. The OpenSSL needs to be configured before executing the “make” command.

Note: The <openssl-arm> and <ssl-arm> are the folders that were created in the previous instruction. The cross-compiler toolchain "arm-linux-gnueabihf-" is used for the Moxa UC-serial computer.

```
$ cd openssl
$ setarch i386 ./config no-asm no-shared enable-ssl3 enable-ssl3-method enable-tls1_3 --prefix=<openssl-arm> --openssldir=<ssl-arm> --cross-compile-prefix=arm-linux-gnueabihf-
```

Copyright © 2021 Moxa Inc.

4. Next, make and install the OpenSSL:

```
$ make
$ make install_sw
```

Finally, the headers and libraries will be constructed in the following hierarchy:

```
openssl-arm
├── bin
├── include
└── lib
    ├── engines
    ├── libcrypto.a
    ├── libssl.a
    └── pkgconfig
```

The following command is to build npreal2d with secure mode:

```
$ arm-linux-gnueabihf-gcc -c ${CFLAGS} -DNO_INIT -DSSL_ON -
DOPENSSL_NO_KRB5 npreal2d.c -I/home/user/openssl-lib/openssl-
arm/include
```

If polling mode is preferred, change "${CFLAGS}" to "-DOFFLINE_POLLING".

```
$ arm-linux-gnueabihf-gcc npreal2d.o -o npreal2d -lssl -lcrypto -
ldl -lpthread -L/home/user/openssl-lib/openssl-arm/lib/ -
I/home/user/openssl-lib/openssl-arm/include
```

The npreal2d binary will be generated.

Note: Only the npreal2d requires OpenSSL library; other binaries should follow Section 2.4.

Note: The secure mode is supported only if the NPort 6000 enables it. Please refer to NPort 6000 Series User Manual to configure secure mode in the NPort 6000.

2.5 Deploy cross-compiled binary to target

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

npreal2.ko
npreal2d
npreal2d_redund
mxloadsvr
mxaddsvr
mxdelsvr

Copyright © 2021 Moxa Inc.

mxsetsec

A few necessary tools are available in the source code directory:

mxcfmat
mxmknod
mxrmnod
npreal2d.cf

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

1. Copy the npreal2.ko to the path /lib/modules/`uname -r`/kernel/drivers/char on the Arm platform.
2. Create a folder /usr/lib/npreal2/driver. Copy all the above files to that folder, except npreal2.ko.
3. Boot into the Arm platform and load the driver.

   ```
   # modprobe npreal2
   ```

4. Change the directory to "/usr/lib/npreal2/driver" and run "mxaddsvr, mxdelsvr, or mxsetsec", the same as running them on x86 Linux.
5. The module can be unloaded by the following command:

   ```
   # modprobe -r npreal2
   ```

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 Real TTY 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 Real TTY 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 the rpi-source to understand the know-how and the limitations.

Copyright © 2021 Moxa Inc.

4 Porting to the Yocto Project on Raspberry Pi

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 Real TTY driver, the Yocto Project, and Raspberry Pi.

The dunfell branch (3.1.9) is referred to throughout in this section. The dunfell branch is for Linux kernel 5.x. If your platform is kernel 4.x, please use zeus branch instead. 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 the 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 YoctoProject is installed in the /home/user/poky folder. Checkout 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'.
4. Add the type 'rpi-sdimg' optionally if SD card is preferred

   IMAGE_FSTYPES="tar.bz2 ext3 rpi-sdimg"

5. 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 ?= "raspberrypi3"

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

Copyright © 2021 Moxa Inc.

7. 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 \
   "
   ```

8. 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.

4.2 Create a Moxa layer for the Yocto Project

4.2.1 Introduction

Moxa RealTTY driver is packaged as a layer for Yocto. You can add or remove the driver by modifying the 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 RealTTY driver.

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

4.2.2 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
   ```

Copyright © 2021 Moxa Inc.

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 anytime.

4.2.3 Create a recipe for the Real TTY kernel

Use the following commands to create a recipe for installing Real TTY 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
```

Copyright © 2021 Moxa Inc.

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

3. Delete the unnecessary files in hello-mod. Rename the hello-mod.

```
$ 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/realtty-kernel_0.1.bb
$ mv ./hello-mod realtty-kernel
```

4. Extract the Real TTY source code in /moxa. Copy the following files into hello-mod:

```
$ cp /moxa/COPYING-GPL.TXT ./realtty-kernel/files/
$ cp /moxa/npreal2.c ./realtty-kernel/files/
$ cp /moxa/npreal2.h ./realtty-kernel/files/
$ cp /moxa/np_ver.h ./realtty-kernel/files/
```

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

```
$ tree ./
./
└── realtty-kernel
    ├── files
    │   ├── COPYING-GPL.TXT
    │   ├── Makefile
    │   ├── npreal2.c
    │   ├── npreal2.h
    │   └── np_ver.h
    └── realtty-kernel_0.1.bb
```

6. Modify the content of the file "./realtty-kernel/files/Makefile" as follows:

```
obj-m := npreal2.o
SRC := $(shell pwd)

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

modules_install:
```

Copyright © 2021 Moxa Inc.

```
$(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
```

7. Modify the content of the file './realtty-kernel/realtty-kernel_0.1.bb' as follows:

```
DESCRIPTION = "Linux kernel module for NPort"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"

inherit module

SRC_URI = " \
file://Makefile \
file://npreal2.h \
file://np_ver.h \
file://npreal2.c \
file://COPYING-GPL.TXT \
"

S = "${WORKDIR}"

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

RPROVIDES_${PN} += "kernel-module-npreal2"
```

4.2.4 Create a recipe for the Real TTY utilities

Similar to creating a realtty-kernel recipe, create a recipe for facilitating the NPort management.

1. Create directory below in meta-moxa:

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

2. Copy the Moxa driver which can be downloaded from the Moxa product web page directly. The driver's name format is npreal2_vM.N_BUILD-DATE.tgz.

```
$ cp /home/user/download/npreal2_vM.N_BUILD_DATE.tgz ./meta-
```

Copyright © 2021 Moxa Inc.

moxa/recipes-utility/realtty-tools/files/

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

```
DESCRIPTION = "Service utilities for NPort"
LICENSE = "GPLv3"
LIC_FILES_CHKSUM = "file://moxa//COPYING-
GPL.TXT;md5=3c34afdc3adf82d2448f12715a255122"

# OpenSSL is required for secured mode
DEPENDS = "openssl"

# Specify the compressed driver file for SRC_URI
SRC_URI = "file://npreal2_vM.N_BUILD-DATE.tgz"

S = "${WORKDIR}"

# Specify the destination of RealTTY driver
DEST_DIR = "${D}${libdir}/npreal2/driver"

FILES_${PN} += "${libdir}/npreal2/driver/*"

# If it is required to connect the NPort with the SSL secure mode (secure
mode is available in the NPort 6000 Series only), unremark the following
line:
#SSL_MODE = "yes"

do_compile () {
${CC} -o mxaddsvr ${S}/moxa/mxaddsvr.c ${S}/moxa/misc.c
${CC} -o mxdelsvr ${S}/moxa/mxdelsvr.c ${S}/moxa/misc.c
${CC} -o mxcfmat ${S}/moxa/mxcfmat.c
${CC} -o mxloadsvr -DNO_INIT ${S}/moxa/mxloadsvr.c ${S}/moxa/misc.c
${CC} -o mxsetsec -DNO_INIT ${S}/moxa/mxsetsec.c ${S}/moxa/misc.c

if [ ${SSL_MODE} = "yes" ], then
${CC} -o npreal2d_redund -lssl -lpthread -DSSL_ON -DOPENSSL_NO_KRB5
${S}/moxa/redund_main.c ${S}/moxa/redund.c
${CC} -o npreal2d -lssl -DSSL_ON -DOPENSSL_NO_KRB5
${S}/moxa/npreal2d.c
or else
${CC} -o npreal2d_redund -lpthread ${S}/moxa/redund_main.c
${S}/moxa/redund.c
${CC} -o npreal2d ${S}/moxa/npreal2d.c
fi
}
```

Copyright © 2021 Moxa Inc.

```
do_install () {
install -m 0755 -d ${DEST_DIR}
install -m 0755 ${S}/mxaddsvr ${DEST_DIR}
install -m 0755 ${S}/mxdelsvr ${DEST_DIR}
install -m 0755 ${S}/mxcfmat ${DEST_DIR}
install -m 0755 ${S}/mxloadsvr ${DEST_DIR}
install -m 0755 ${S}/mxsetsec ${DEST_DIR}
install -m 0755 ${S}/moxa/mxmknod ${DEST_DIR}
install -m 0755 ${S}/moxa/mxrmnod ${DEST_DIR}
install -m 0755 ${S}/npreal2d ${DEST_DIR}
install -m 0755 ${S}/npreal2d_redund ${DEST_DIR}
install -m 0755 ${S}/moxa/npreal2d.cf ${DEST_DIR}
}

# Ignore GNU_HASH (did not pass LDFLAGS)
INSANE_SKIP_${PN} = "ldflags"
```

Note: The file name of SRC_URI must be the same as it was copied in the last step.

4. The content of meta-moxa is listed as below:

```
$ tree meta-moxa
meta-moxa
├── conf
│   └── layer.conf
├── COPYING.MIT
├── README
├── recipes-kernel
│   └── realtty-kernel
│       ├── files
│       │   ├── COPYING-GPL.TXT
│       │   ├── Makefile
│       │   ├── npreal2.c
│       │   ├── npreal2.h
│       │   └── np_ver.h
│       └── realtty-kernel_0.1.bb
└── recipes-utility
    └── realtty-tools
        ├── files
        │   └── npreal2_vM.N_BUILD-DATE.tgz
        └── realtty-tools.bb
```

Copyright © 2021 Moxa Inc.

4.3 Install a Moxa layer into the Yocto Project

1. Install the Moxa layer and Real TTY recipes into the Yocto Project.

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

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

3. 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 \
   "

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

   IMAGE_INSTALL_append += " realtty-tools realtty-kernel"

4.4 Deploy the Yocto image in Raspberry Pi

Build the image with the Real TTY driver:

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

An SD-card format image (.rpi-sdimg) is generated under /home/user/poky/build/tmp/deploy/images/raspberrypi3. 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.

Copyright © 2021 Moxa Inc.

4.5 Start the Real TTY driver in Raspberry Pi

After logging into the system, start the Real TTY driver

```
root@raspberrypi3:~# modprobe npreal2
[    39.906812] npreal2: loading out-of-tree module taints kernel.
[    39.913379] MOXA Async/NPort server family Real TTY driver ttymajor 33
calloutmajor 38 verbose 1 (Ver5.1)
```

For example, we illustrate how to add a 4-port NPort with the IP address: 192.168.127.254

```
root@raspberrypi3:~# cd /usr/lib/npreal2/driver
root@raspberrypi3:/usr/lib/npreal2/driver# ./mxaddsvr 192.168.127.254 4
Adding Server...

ttyr00, cur00
ttyr01, cur01
ttyr02, cur02
ttyr03, cur03
Added Real Com IP : 192.168.127.254
```

Now the device node /dev/ttyr00 ~ /dev/ttyr03 is created for tty port use.

4.6 Set the default tty mapping to the Real TTY configuration

You may use the Real TTY configuration file, npreal2d.cf that we set up in 4.5, as the default settings when deploying to a new Raspberry Pi image.

1. Copy and replace npreal2d.cf in the NPort Real TTY driver folder '/moxa' extracted in the build system.
2. tar -zxvf new_npreal2_driver.tgz /moxa
3. Go back to 4.2.4, change the name of npreal2_vM.N_BUILD_DATE.tgz with the file name in step 2.)
4. Rebuild the image.

Copyright © 2021 Moxa Inc.

4.7 (Optional): Use the SSL secure mode for the NPort 6000 Series

You may use the NPort secure mode (SSL) to connect between Raspberry Pi and the NPort 6000 Series securely. The following instructions are for this purpose:

1. Open the realtty-tools bb file with a text editor.

   (./meta-moxa/recipes-utility/realtty-tools/realtty-tools.bb)

2. If it is required to connect the NPort with the SSL secure mode (secure mode is available in the NPort 6000 Series only), unremark the following line: SSL_MODE = "yes"
3. Repeat 4.4 and 4.5 again, executing the following command to enable the serial port after the NPort mapping. Remember to enable secure mode in the NPort.

   ```
   root@raspberrypi3:/usr/lib/npreal2/driver# ./mxsetsec
   ```

4.8 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
```

Copyright © 2021 Moxa Inc.