KRIA board Ubuntu LTS 22.04 Install

Vincent Conus*

2023-12-22

Contents

1 Preparing and booting a Ubuntu 22.04 media 2

2 Network and admin setups 2
2.1 Static IP address . . . 3
2.2 [facultative] Proxy for `apt` on Nagoya University network . . . 3
2.3 [facultative] Check the system clock . . . 3
2.4 [facultative] `root` password . . . 3
2.5 Adding Xilinx specific repositories . . . 4
2.6 [facultative] Installing Docker . . . 4
2.7 [recommended] Purging `snap` . . . 5
2.8 [facultative] Other unused heavy packages . . . 6
2.9 [facultative] Slow boot services to disable . . . 6
2.10 [facultative] Adding a swap partition . . . 6
2.11 [OLD] Enabling `remoteproc` with Device-Tree patching . . . 7
2.11.1 Patching the device tree for RPMsg (standard, kernel space mode) . . . 7
2.11.2 Kernel `cmd` edit . . . 7
2.11.3 Checking the patching . . . 8
2.12 [NEW] Enabling `remoteproc` for RPMsg in userspace with device-tree patch . . . 9
2.12.1 Kernel `cmd` edit . . . 9
2.12.2 Patching the device tree for RPMsg in userspace . . . 9
2.12.3 Checking the patching . . . 9
2.13 [NEW] Nagoya University proxy and NTP . . . 9
2.13.1 General HTTP proxy . . . 9
2.13.2 HTTP proxy for the package manager . . . 10
2.13.3 Local NTP server . . . 10
2.13.4 Git forges SSH access through proxy . . . 10
2.14 [DEPRECATED] Nanzan Net Proxy and DNS . . . 11
2.15 [DEPRECATED] Jupyter notebook setup . . . 11
2.16 [TODO] Using a PetaLinux kernel in Ubuntu . . . 12

3 RPMsg standalone evaluation 12
3.1 [TODO] RPMsg Cortex R5F demonstration firmware . . . 12
3.2 RPMsg `echo_test` software . . . 12

*vincent.conus@protonmail.com

1 Preparing and booting a Ubuntu 22.04 media

An official Ubuntu image exists and is provided by Xilinx, allowing the OS installation to be quick and straightforward. Ubuntu is a common and easy to use distribution. Furthermore, it allows to install ROS2 as a package, which is most convenient and will be done later in this guide.

Once the image has been downloaded at Canonical's page we can flash it onto the SD card, with the following instructions.

DANGER: The next part involve the dd command writing on disks!!! As always with the dd command, thou have to be **VERY** careful on what arguments thou give. Selecting the wrong disk will result on the destruction of thy data !! If you are unsure of what to do, seek assistance !

With the image available on thy machine and a SD card visible as /dev/sda device[1] one can simply run the dd command as follow to write the image to a previously formatted drive (here /dev/sda):

```
unxz iot-limerick-kria-classic-desktop-2204-20240304-165.img.xz
sudo dd if=iot-limerick-kria-classic-desktop-2204-20240304-165.img \
     of=/dev/sda status=progress bs=8M && sync
```

Once the SD card is flashed and put back in the board, the micro-USB cable can be connected from the PC to the board. It is then possible to connect to the board in serial with an appropriate tool, for example picocom, as in the following example (the serial port that "appeared" was the /dev/ttyUSB1 in this case[2], and the 115200 bit-rate is the default value for the board):

```
sudo picocom /dev/ttyUSB1 -b 115200
```

In my case, I am using Emacs's serial-term:

```
M-x serial-term RET /dev/ttyUSB1 RET 115200 RET
```

The default username / password pair for the very first boot is ubuntu and ubuntu. You will then be prompted to enter a new password.

Once logged in, it is typically easier and more convenient to connect the board using SSH. When the board is connected to the network, it is possible to know it's IP address with the IP command; then it is possible to connect to the board with ssh, as follow (example, with the first command to be run on the board and the second one on the host PC, both without the first placeholder hostnames):

```
kria# ip addr

host# ssh ubuntu@192.168.4.11
```

2 Network and admin setups

This section presents a variety of extra convenience configurations that can be used when setting-up the Kria board.

[1]Again, it is critical to be 100\the correct device!

[2]If two boards are plugged in for serial, the second one will be /dev/ttyUSB5, then USB9 and so on.

2.1 Static IP address

A static IP can be set by writing the following configuration into your `netplan` configuration file[3].
The name of the files might vary:

```
sudo chmod 0600 /etc/netplan/50-cloud-init.yaml
sudo nano /etc/netplan/50-cloud-init.yaml
```

You can then set the wanted IP as follow[4]:

```
network:
  renderer: NetworkManager
  version: 2
  ethernets:
    eth0:
      dhcp4: false
      addresses:
        - 192.168.11.107/24
      routes:
        - to: default
          via: 192.168.11.1
      nameservers:
        addresses: [192.168.11.1]
```

Finally, the change in settings can be applied as follow:

```
sudo netplan apply
```

2.2 [facultative] Proxy for `apt` on Nagoya University network

This is a configuration specific to use the board at Nagoya University, on the local network. The following configuration is to be used in `/etc/apt/apt.conf`:

```
Acquire::http::Proxy "http://172.24.8.1:8080";
Acquire::https::Proxy "http://172.24.8.1:8080";
```

2.3 [facultative] Check the system clock

2.4 [facultative] `root` password

WARNING: Depending on your use-case, the setup presented in this subsection can be a critical security breach as it remove the need for a root password to access the admin functions of the board's Linux. When in doubt, do not apply this configuration!!

If you board does not hold important data and is available to you only, for test or development, it might be convenient for the `sudo` tool to not ask for the password all the time. This change can be done by editing the sudoers file, and adding the parameter `NOPASSWD` at the `sudo` line:

```
sudo visudo

%sudo   ALL=(ALL:ALL) NOPASSWD: ALL
```

[3]The `chmod` command is used to update the permissions and silence some warnings
[4]For the routing part, it is key to have the `to` with a `'-'` in front of it; and then the `via` without, but aligned with the `t`.

Again, this is merely a convenience setup for devices staying at you desk. If the board is meant to be used in any kind of production setup, a password should be set for making administration tasks.

With all of these settings, you should be able to update the software of your board without any issues:

```
sudo apt update
sudo apt dist-upgrade
sudo reboot now
```

2.5 Adding Xilinx specific repositories

The following commands will add PPA repositories that are specific for Xilinx boards using Ubuntu. It is then possible to update the package list and eventually upgrade to some new packages.

```
sudo add-apt-repository ppa:ubuntu-xilinx/updates
sudo add-apt-repository ppa:xilinx-apps/ppa
sudo apt update
sudo apt upgrade
```

2.6 [facultative] Installing Docker

It is possible to have a version of Docker installed simply by using the available repository, but since we are on Ubuntu, a PPA is available from Docker in order to have the most up-to-date version.

Following the official documentation, the following steps can be taken to install the latest version of Docker on a Ubuntu system. The last command is meant to test the install. If everything went smoothly, you should see something similar to what is presented in the figure 1 below, after the commands:

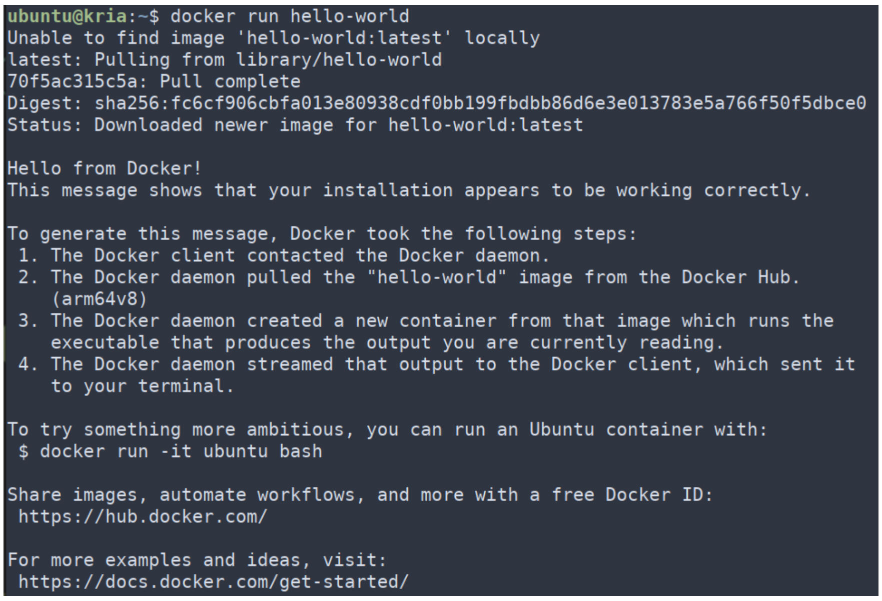
```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
    sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

sudo chmod a+r /etc/apt/keyrings/docker.gpg

echo \
    "deb [arch="$(dpkg --print-architecture)" \
  signed-by=/etc/apt/keyrings/docker.gpg] \
  https://download.docker.com/linux/ubuntu \
  "$(. /etc/os-release && \
         echo "$VERSION_CODENAME")" stable" | \
    sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli \
     containerd.io docker-buildx-plugin docker-compose-plugin
sudo usermod -aG docker $USER
newgrp docker

docker run hello-world
```



```
ubuntu@kria:~$ docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
70f5ac315c5a: Pull complete
Digest: sha256:fc6cf906cbfa013e80938cdf0bb199fbdbb86d6e3e013783e5a766f50f5dbce0
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (arm64v8)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
 $ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
 https://hub.docker.com/

For more examples and ideas, visit:
 https://docs.docker.com/get-started/
```

Figure 1: The return of a successful run of the `hello world` test Docker container.

2.7 [recommended] Purging `snap`

As the desktop-specific software are not used at all in the case of our project, there are some packages that can be purges in order for the system to become more lightweight.

In particular, the main issue with Ubuntu systems is the forced integration of Snap packages. Here are the command to use in order to remove all of that. These steps take a lot of time and need to be executed in that specific order[5], but the system fan runs sensibly slower without all of this stuff.

Note that this all process is rather slow and can take up to 30min to complete.

```
sudo systemctl disable snapd.service
sudo systemctl disable snapd.socket
sudo systemctl disable snapd.seeded.service

sudo snap list #show installed package, remove them all:
sudo snap remove --purge firefox
sudo snap remove --purge gnome-3-38-2004
sudo snap remove --purge gnome-42-2204
sudo snap remove --purge gtk-common-themes
sudo snap remove --purge snapd-desktop-integration
sudo snap remove --purge snap-store
sudo snap remove --purge bare
sudo snap remove --purge core20
sudo snap remove --purge core22
sudo snap remove --purge snapd
sudo snap list # check that everything is uninstalled

sudo rm -rf /var/cache/snapd/
sudo rm -rf ~/snap
sudo apt autoremove --purge snapd

# check once more that there is no more snap on the system
```

[5]The snap packages depends on each others. Dependencies cannot be remove before the package(s) that depends on them, thus the specific delete order.

```
systemctl list-units | grep snapd
```

2.8 [facultative] Other unused heavy packages

Some other pieces of software can safely be removed since the desktop is not to be used.

```
sudo apt-get autoremove --purge yaru-theme-icon \
     fonts-noto-cjk yaru-theme-gtk vim-runtime \
     ubuntu-wallpapers-jammy humanity-icon-theme

sudo apt-get autoclean
sudo reboot now
```

2.9 [facultative] Slow boot services to disable

These packages (in particular the first one) are taking up a LOT of time at boot while providing no benefits[6].
It is possible to disable them as follow:

```
sudo systemctl disable systemd-networkd-wait-online.service
sudo systemctl disable NetworkManager-wait-online.service
sudo systemctl disable cups.service
sudo systemctl disable docker.service
sudo systemctl disable containerd.service
sudo systemctl disable cloud-init-local.service
```

Additional, potentially unused services can be found using the very handy command:

```
sudo systemd-analyze blame
```

2.10 [facultative] Adding a swap partition

This part is very optional, in particular as it might slow down a bit the boot time of the board (~2s), however it might become handy to have swap memory available to avoid system failure under heavy use.
This whole procedure must be done externally, with the board system SD card mounted on a host PC as an external volume. As it is highly platform dependant, I will not give a detailed explanation on how to do it, yet here are the key points that should be done:

- Shutdown the Kria board, take out the SD card and put it in a host machine.
- Make sure the disk is visible.
- Make sure all volumes are **unmounted**.
- Resize the main `root` partition (**not** the `boot`) so a space the size of the wanted swap is free **after** the partition. You'd want something around 1GB.
- In the empty space, create a new partition, which type is "linux swap".
- Find and take note of the UUID of the new partition. This is useful hereafter.
- `sync`
- Un-mount everything, eject SD card.

[6]The CUPS and Docker services will be activated when used instead of during boot time.

- Put the SD card back in the Kria.
- Boot back to Ubuntu.

Going back on the Kria board Ubuntu after boot, the `/etc/fstab` file can be updated as follow, modulo your actual UUID for the newly created partition, to enable swap at boot time.

```
sudo -s
echo "UUID=8b13ed05-a91d-4x50-a44a-e654a0c67a2c none   swap    sw      0       0" >> /etc/fstab
reboot now
```

2.11 [OLD] Enabling `remoteproc` with Device-Tree patching

One of the advantage of this Kria board, as cited previously, is the presence of multiple types of core (APU, MCU, FPGA) on the same chip.

The part in focus in this guide is the usage of both the APU, running a Linux distribution and ROS2; and the MCU, running FreeRTOS and micro-ROS. Online available guides[7] , [8] also provide information on how to deploy these types of systems and enabling `remoteproc` for the Kria board, but this guide will show a step-by-step, tried process to have a heterogeneous system up and running.

The communication between both side is meant to be done using shared memory, but some extra setup is required in order to be running the real-time firmware, in particular for deploying micro-ROS on it.

As a first step in that direction, this section of the report will present how to setup and use as an example firmware that utilizes the `remoteproc` device in Linux in order to access shared memory and communicate with the real-time firmware using the RPMsg system.

The communication system and interaction from the Linux side towards the real-time capable core is not enabled by default within the Ubuntu image provided by Xilinx.

In that regard, some modification of the device tree overlay (DTO) is required in order to have the `remoteproc` system starting.

2.11.1 Patching the device tree for RPMsg (standard, kernel space mode)

Firstly, we need to get the original firmware device tree, converted into a readable format (DTS):

```
sudo dtc /sys/firmware/fdt 2> /dev/null > system.dts
```

Then, a custom-made patch file can be downloaded and applied. This file is available at the URL visible in the command below.

```
wget https://gitlab.com/sunoc/xilinx-kria-kv260-documentation/-/raw/main/src/system.patch

patch system.dts < system.patch
```

2.11.2 Kernel `cmd` edit

As for the board to be able to reserve the correct amount of memory with the new settings, some `cma` kernel configuration is needed[9]:

[7]A slideshow (JP) from Fixstar employees presents how to use the device tree to enable the communication between the cores.
[8]A blog post (JP) shows all major steps on how to enable the `remoteproc`.
[9]The overlapping memory will not prevent the board to boot, but it disables the PWM for the CPU fan, which will then run at full speed, making noise.

```
sudo nano /etc/default/flash-kernel

LINUX_KERNEL_CMDLINE="quiet splash cma=512M cpuidle.off=1"
LINUX_KERNEL_CMDLINE_DEFAULTS=""
sudo flash-kernel
```

Now the DTS file has been modified, one can regenerate the binary and place it on the `/boot` partition and reboot the board:

```
dtc -I dts -O dtb system.dts -o user-override.dtb
sudo mv user-override.dtb /boot/firmware/
sudo reboot now
```

2.11.3 Checking the patching

After rebooting, you can check the content of the `remoteproc` system directory, and a `remoteproc0` device should be visible, as follow:

```
ls /sys/class/remoteproc/
#  remoteproc0
```

If it is the case, it means that the patch was successful and that the remote processor is ready to be used!

2.12 [NEW] Enabling `remoteproc` for RPMsg in userspace with device-tree patch

2.12.1 Kernel `cmd` edit

```
sudo nano /etc/default/flash-kernel

LINUX_KERNEL_CMDLINE="quiet splash cma=512M cpuidle.off=1"
LINUX_KERNEL_CMDLINE_DEFAULTS=""
sudo flash-kernel
```

2.12.2 Patching the device tree for RPMsg in userspace

The `system_uio.patch`

```
sudo dtc /sys/firmware/fdt 2> /dev/null > system.dts

wget https://gitlab.com/sunoc/xilinx-kria-kv260-documentation/-/raw/main/src/system_uio.patch

patch system.dts < system_uio.patch

dtc -I dts -O dtb system.dts -o user-override.dtb
sudo mv user-override.dtb /boot/firmware/
sudo reboot now
```

2.12.3 Checking the patching

If everything went correctly, on reboot and re-logging in the device, you should see an output to the following command:

```
ls /sys/class/remoteproc/
```

2.13 [NEW] Nagoya University proxy and NTP

WARNING: This setup is needed when using the KRIA board with wired internet at Honda Lab in Nagoya University, because of the local network proxy.

Two main addresses for working behind the lab's local network proxy are:

- `172.24.8.1` for HTTP and HTTPS proxy.
- `172.24.8.19` for the local NTP server.

The following sub-sections will present how to use these.

2.13.1 General HTTP proxy

These lines must be places at the end of the `~/.bashrc` file:

```
export http_proxy="172.24.8.1:8080"
export https_proxy="172.24.8.1:8080"
export no_proxy="localhost,127.0.0.1,::1"
```

2.13.2 HTTP proxy for the package manager

These lines must be places at the end of the `/etc/apt/apt.conf` file:

```
Acquire::http::Proxy "http://172.24.8.1:8080";
Acquire::https::Proxy "http://172.24.8.1:8080";
```

2.13.3 Local NTP server

Installing a NTP tool:

```
sudo apt update
sudo apt install systemd-timesyncd
```

These lines must be places at the end of the `/etc/systemd/timesyncd.conf` file:

```
[Time]
NTP=172.24.8.19
```

Start and enable the NTP tool systemd daemon:

```
sudo systemctl start systemd-timesyncd
sudo systemctl enable systemd-timesyncd
timedatectl
```

2.13.4 Git forges SSH access through proxy

A SSH through HTTP tunneling tool must be installed:

```
sudo apt update
sudo apt install corkscrew
```

These lines must be places at the end of the `~/.gitconfig` file:

```
Host github.com
     User git
     Port 443
     Hostname ssh.github.com
     TCPKeepAlive yes
     IdentityFile ~/.ssh/id_ed25519
     IdentitiesOnly yes
     ProxyCommand /usr/bin/corkscrew 172.24.8.1 8080 %h %p

Host gitlab.com
     User git
     Port 443
     Hostname altssh.gitlab.com
     IdentityFile ~/.ssh/id_ed25519
     ProxyCommand /usr/bin/corkscrew 172.24.8.1 8080 %h %p
```

2.14 [DEPRECATED] Nanzan Net Proxy and DNS

WARNING: This setup is not needed anymore when using Honda-sensei's lab wired network. Adding it will cause the DNS to FAIL! This section is kept as a reference.

An issue that can occur when connecting the board to the internet is the conflicting situation with the university proxy. Indeed, as the network at Nanzan University requires to go through a proxy, some DNS errors appeared.

In that case, it might become needed to setup the proxy for the school.

This can be done as follow, by exporting a https base proxy configuration containing you AXIA credentials (this is specific to Nanzan University IT system), then by consolidating the configuration for other types of connections in the `bashrc`:

```
export https_proxy="http://<AXIA_username>:\
       <AXIA_psw>@proxy.ic.nanzan-u.ac.jp:8080"

echo "export http_proxy=\""$https_proxy"\"" >> ~/.bashrc
echo "export https_proxy=\""$https_proxy"\"" >> ~/.bashrc
echo "export ftp_proxy=\""$https_proxy"\"" >> ~/.bashrc
echo "export no_proxy=\"localhost, 127.0.0.1,::1\"" >> ~/.bashrc
```

Eventually the board can be rebooted in order for the setup to get applied cleanly.

2.15 [DEPRECATED] Jupyter notebook setup

Here are some instruction on how to install and setup Jupyter on a KRIA board, accessing it remotely and using it for making data analysis.

The following commands will set the required packages and install Jupyter itself[10]:

```
sudo apt-get update && sudo apt-get install python3 python3-pip python3-venv python3-virtualenv

virtualenv myjupyter
source ./myjupyter/bin/activate
python3 -m pip install jupyter pandas numpy matplotlib scipy

sudo reboot now
```

Then in a terminal on your host machine (not on the KRIA board), you can run the following command[11] to bind local ports:

```
ssh -L 8888:localhost:8888 ubuntu@192.168.11.107
```

Then on the opened SSH shell to the KRIA board:

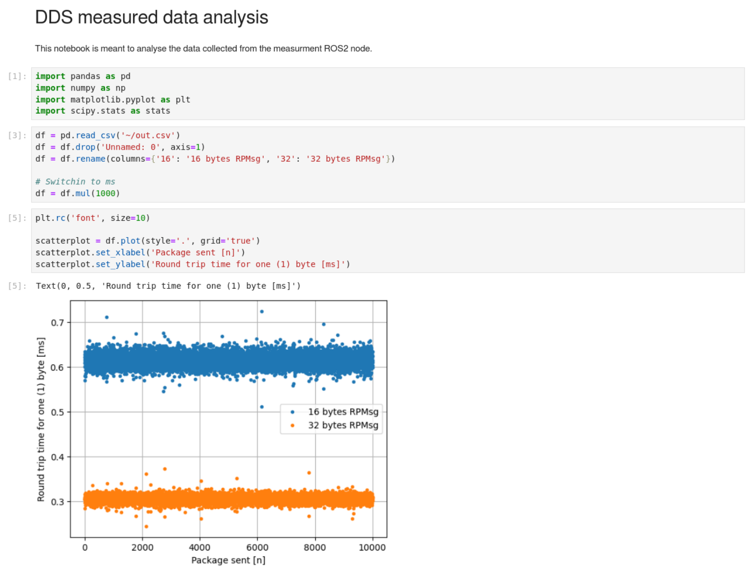
```
source ./myjupyter/bin/activate
jupyter notebook
```

From there, it is possible to use the displayed URL (something that looks like `http://localhost:8888/tree?token`) to access the remote Notebook system from a local web browser. It is possible to do so with `localhost` since we have the `ssh` port map connection going on.

Eventually creating Notebooks and stuff, it is possible to obtain a situation like shown in the figure 2 below.

[10] Alongside other packages useful for data analysis, such as `pandas` or `numpy`.

[11] In this example, the full `username@IP` is used, but a `.ssh/config` is also usable.


DDS measured data analysis

This notebook is meant to analyse the data collected from the measurment ROS2 node.

```
[1]: import pandas as pd
     import numpy as np
     import matplotlib.pyplot as plt
     import scipy.stats as stats

[3]: df = pd.read_csv('~/out.csv')
     df = df.drop('Unnamed: 0', axis=1)
     df = df.rename(columns={'16': '16 bytes RPMsg', '32': '32 bytes RPMsg'})

     # Switchin to ms
     df = df.mul(1000)

[5]: plt.rc('font', size=10)

     scatterplot = df.plot(style='.', grid='true')
     scatterplot.set_xlabel('Package sent [n]')
     scatterplot.set_ylabel('Round trip time for one (1) byte [ms]')

[5]: Text(0, 0.5, 'Round trip time for one (1) byte [ms]')
```

Figure 2: A test Jupyter Notebook for CSV data analysis.

2.16 [TODO] Using a PetaLinux kernel in Ubuntu

TBD

3 RPMsg standalone evaluation

3.1 [TODO] RPMsg Cortex R5F demonstration firmware

3.2 RPMsg `echo_test` software

In order to test the deployment of the firmware on the R5F side, and in particular to test the RPMsg function, we need some program on the Linux side of the KRIA board to "talk" with the real-time side.

Some source is provided by Xilinx to build a demonstration software that does this purpose: specifically interact with the demonstration firmware.

Here are the steps required to obtain the sources, and build the program.

As a reminder, this is meant to be done on the Linux running on the KRIA board, NOT on your host machine !

```
git clone https://github.com/Xilinx/meta-openamp.git
cd  meta-openamp
git checkout xlnx-rel-v2022.2
cd  ./recipes-openamp/rpmsg-examples/rpmsg-echo-test
make
sudo ln -s $(pwd)/echo_test /usr/bin/
```

Once this is done, it it possible to run the test program from the KRIA board's Ubuntu by running the `echo_test` command.