Setting up and using Xilinx's KRIA KV260 board

南山大学

Vincent Conus - Source available at GitLab

Contents

1 Introduction and motivation

1.1 Documentation and guides

- Board product information:
  `https://www.xilinx.com/products/som/kria/kv260-vision-starter-kit.html`
- SoC product information:
  `https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html`
- Download page for the official Canonical Ubuntu releases for the board:
  `https://ubuntu.com/download/amd-xilinx`
- Xilinx official documentation:
  `https://docs.xilinx.com/r/en-US/ug1089-kv260-starter-kit/Summary`
- Atlassian documentation:
  `https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/1641152513/Kria+K26+SOM`
- Fixstar presentation (JP):
  `https://speakerdeck.com/fixstars/fpga-seminar-12-fixstars-corporation-20220727`
- Xilinx OpenAMP demo documentation:
  `https://xilinx.github.io/kria-apps-docs/openamp/build/html/openamp_landing.html`
- Libmetal and OpenAMP official Xilinx documentation:
  `https://docs.xilinx.com/r/en-US/ug1186-zynq-openamp-gsg/Introduction?tocId=GFruK4_sY1eyu3jD9X1EuA`
- Guide for KV260 board setup (JP):
  `https://zenn.dev/ryuz88/articles/kv260_setup_memo_ubuntu22`
- Vitis IDE download page:
  `https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html`

2 Boot Image Update

In order to be able to boot a newer version of Linux, the boot image of the board must be updated.
The procedure is available in the official documentation, but I will present it step by step here.

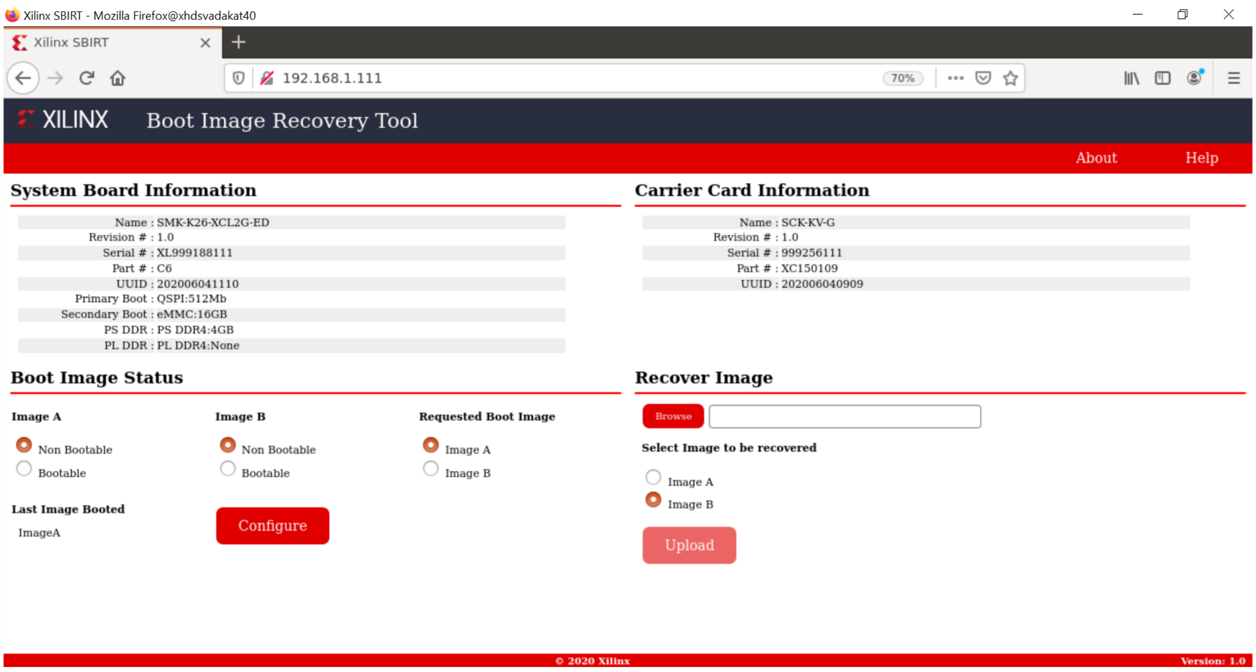
2.1 Getting the new firmware

A 2022 version of the board firmware is required in order to run the latest version of Ubuntu properly.
The image can be downloaded at the atlassian page on the topic.

2.2 Reaching the board recovery tool

Now the firmware image is available, it is possible to update it using the boards recovery tool. Here are the steps that must be taken in order to reach this tool and update the board:

- Connect the board to your machine via Ethernet.
- Select the wired network as your connection (must be "forced", since it doesn't have internet access).
- Set a fixed IP address for your machine, in the `192.168.0.1/24` range, except the specific `192.168.0.111`, which will be used by the board.
- Using a web browser on your host machine, access `http://192.168.0.111`. Thou shall now see the interface, as visible on the figure 1 below.
- Now the downloaded file can be uploaded as a recovery image in the bottom-right section.



Boot Image Recovery Tool

System Board Information

Name : SMK-K26-XCL2G-ED
Revision # : 1.0
Serial # : XL999188111
Part # : C6
UUID : 202006041110
Primary Boot : QSPI:512Mb
Secondary Boot : eMMC:16GB
PS DDR : PS DDR4:4GB
PL DDR : PL DDR4:None

Carrier Card Information

Name : SCK-KV-G
Revision # : 1.0
Serial # : 999256111
Part # : XC150109
UUID : 202006040909

Boot Image Status

Image A
[✓ Non Bootable] Bootable

Image B
[✓ Non Bootable] Bootable

Requested Boot Image
[✓ Image A] Image B

Last Image Booted
ImageA

Recover Image

Select Image to be recovered
Image A [✓ Image B]

© 2020 Xilinx
Version: 1.0

Figure 1: The recovery tool for the board, access from Firefox. We can see board information at the center, and the tools to upload the firmware at the bottom of the page.

3 Ubuntu 22.04

An official Ubuntu image is provided by Xilinx that allows 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.

One the image has been downloaded at Canonical's page (`https://ubuntu.com/download/amd-xilinx`), we can flash it onto the SD card.

WARNING: 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 thou art unsure of what to do, seek assistance !

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

```
sudo dd if=iot-limerick-kria-classic-desktop-2204-x07-20230302-63.img
of=/dev/sda status=progress bs=8M && sync
```

Once the SD card was flash, put back on the board and a micro-USB cable has been connected from the PC to the board, it is possible to connect to the board in serial with an appropriate tool, for example picocom, as in the following example:

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

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):

```
kria# ip addr

host# ssh ubuntu@192.168.4.11
```

3.1 Proxy, DNS and root password

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

Firstly, it is possible to set a DNS IP address in `/etc/resolv.conf` by editing it and adding your favorite DNS, for example `nameserver 1.1.1.1` next to the other `nameserver` entry.

```
sudo nano /etc/resolv.conf

sudo systemctl restart systemd-resolved
```

Secondly, 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, then by consolidating the configuration for other types of connections in the `bashrc`:

[1] Again, it is critical to be 100% certain that you are working with the correct device!

```
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
```

Finally, 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
```

This is merely a convenience setup for devices staying at you desk. If the board is meant to be used in any king 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-get update
sudo apt-get dist-upgrade
sudo reboot now
```

3.2 Setting a static IP address

A static IP can be set by writing the following configuration into your `netplan` configuration file.

The name of the files might vary:

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

You can then set the wanted IP as follow:

```
network:
  renderer: NetworkManager
  version: 2
  ethernets:
    eth0:
      addresses:
        - 192.168.11.103/24
      routes:
        - to: default
          via: 192.168.11.1
      nameservers:
        addresses:
          - 8.8.8.8
          - 1.1.1.1
```

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

```
sudo netplan apply
```

3.3 Extra: removing useless stuff

As the desktop part is not used at all, 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. Here are some step to try and remove all of that. These steps take a lot of time though, but the system fan runs sensibly slower without all of this stuff:

```
sudo systemctl disable snapd.service
sudo systemctl disable snapd.socket
sudo systemctl disable snapd.seeded.service
sudo snap list #show installed package, remove then 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

systemctl list-units | grep snapd
```

And here are some other stuff that can safely be removed too is 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
```

4 Starting remoteproc

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

The part we are focusing in this guide is the usage of both the APU, running a Linux distribution and ROS2; and the MCU, running FreeRTOS and micro-ROS.

The communication between both side is meant to be done using shared memory, but some extra setup is required. This section will present how to setup and use as an example the `remoteproc` and RPMsg system.

4.1 References

This exact use-case is not exactly presented in the official documentation, but some guides were found to eventually put together this procedure.

- A slideshow (JP) from Fixstar employees presents how to use the device tree to enable the communication between the cores.
  `https://speakerdeck.com/fixstars/fpga-seminar-12-fixstars-corporation-20220727`
- This blog article (JP) shows all major steps on how to enable the `remoteproc`.
  `https://zenn.dev/ryuz88/articles/kv260_setup_memo_ubuntu22`

4.2 DTO patching

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.

Because of that, some modification of the device tree is required in order to have the `remoteproc` system to start.

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 we can download and apply the patch[2] that will allow us to use the `remoteproc`:

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

patch system.dts < system.patch
```

As for the board to be able to reserve the correct amount of memory with the new settings, some `cma` kernel configuration is needed[3]:

```
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:

[2] The file is also visible in the appendix A at the end of this report.

[3] 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.

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

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 RPMsg is ready to test with some example; however, in order for these examples to be built, we need to setup Xilinx preferred IDE: Vitis.

5 Building and running an example program on the Linux side

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/
```

6 Building the micro-ROS static library

In this section, the goal is to build the micro-ROS library in order to be able to integrate the micro-ROS function into our Cortex R5F firmware. All of this should be done via cross-compiling on a host machine, however it is most common in the guides about micro-ROS to build the firmwares and libraries within a Docker, so we can have access of the ROS environment without installing it on my machine. One can simply run this command to summon a ROS2 Docker[4] with the wanted version, but first we also need to check the cross-compilation tools.

We are downloading the latest `arm-none-eabi` gcc compiler directly
from the ARM website. The cross-compilation tool can then be extracted, set as our "toolchain" variable, then passed as a parameter when creating the Docker container. Here is how to do all of this and then opening a `bash` shell in the container:

[4]If Docker is not set up on your machine, you can follow the guide on the official website. When you can successfully run the "hello-world" container, you are good to go.

```
pushd /home/$USER/Downloads
wget https://developer.arm.com/-/media/Files/downloads/\
gnu/12.2.mpacbti-rel1/binrel/arm-gnu-toolchain-12.2\
.mpacbti-rel1-x86_64-arm-none-eabi.tar.xz
tar -xvf arm-gnu-toolchain-12.2.mpacbti-rel1-x86_64-\
arm-none-eabi.tar.xz
popd

toolchain="/home/$USER/Downloads/arm-gnu-toolchain-\
12.2.mpacbti-rel1-x86_64-arm-none-eabi/"


docker run -d --name ros_build -it --net=host \
--hostname ros_build \
-v /dev:/dev \
-v $toolchain:/armr5-toolchain \
--privileged ros:iron

docker exec -it ros_build bash
```

Now we are in the ROS2 container, we can build the micro-ROS firmware as presented in the dedicated micro-ROS guide:

```
sudo apt update && \
sudo apt-get -y install python3-pip \
                        wget \
                        nano
```

```
. /opt/ros/$ROS_DISTRO/setup.bash

mkdir microros_ws
cd microros_ws
git clone -b $ROS_DISTRO \
https://github.com/micro-ROS/micro_ros_setup.git \
src/micro_ros_setup

sudo rosdep fix-permissions &&\
rosdep update &&\
rosdep install --from-paths src --ignore-src -y

colcon build
. ./install/local_setup.bash

ros2 run micro_ros_setup create_firmware_ws.sh generate_lib
```

From there we will need some configuration files specifically for our Cortex R5F. Both configuration files[5] will be downloaded from my repository; we also are going to copy the cross-compiler into the microros workspace, then we can build the library with the following ros2 command:

[5]Both files are also visible in the appendixes B and C at the end of this report.

```
wget https://gitlab.com/sunoc/xilinx-kria-kv260-\
documentation/-/raw/main/src/custom_r5f_toolchain.cmake

wget https://gitlab.com/sunoc/xilinx-kria-kv260-\
documentation/-/raw/main/src/custom_r5f_colcon.meta

cp -r /armr5-toolchain/ $(pwd)/firmware/toolchain && \
export PATH=$PATH:$(pwd)/firmware/toolchain/bin

ros2 run micro_ros_setup build_firmware.sh \
$(pwd)/custom_r5f_toolchain.cmake $(pwd)/\
custom_r5f_colcon.meta

exit
```

7 Vitis IDE

This is the recommended IDE used to build software for the Xilinx boards. It also include the tools to interact with the FPGA part, making the whole software very large (around 200GB of disk usage).

The installer can be found on Xilinx download page. You will need to get a file named something like "Xilinx_Unified_2022.2_1014_8888_Lin64.bin".

Vitis IDE installer is compatible with versions of Ubuntu, among other distributions, but not officially yet for the 22.04 version. Furthermore, the current install was tested on Pop OS, a distribution derived from Ubuntu.

This guide will present a guide that supposedly works for all distributions based on the newest LTS from Ubuntu. For other Linux distributions or operating system, please refer to the official documentation.

7.1 Install dependencies

Some packages are required to be installed in order for the installation process to happen successfully:

```
sudo apt-get update
sudo apt-get install libncurses-dev \
ncurses-term \
ncurses-base \
ncurses-bin \
libncurses5 \
libtinfo5 \
libncurses5-dev \
libncursesw5-dev
```

Once this is done, the downloaded binary installer can be executed:

```
./Xilinx_Unified_2022.2_1014_8888_Lin64.bin
```

If it is not possible to execute it, make it executable with the `chmod` command:

```
sudo chmod +x ./Xilinx_Unified_2022.2_1014_8888_Lin64.bin
```

From there you can follow the step-by-step graphical installer. The directory chosen for the rest of this guide for the `Xilinx` directory is directly the `\$HOME`, but the installation can be set elsewhere is needed.

8 Building an example firmware for the Cortex R5F

As visible on the official Xilinx documentation about building a demo firmware, this section will present the required steps for building a new firmware for the R5F core of our Kria board. The goal here is to have a demonstration firmware running, able to use the RPMsg system to communicate with the Linux APU.

8.1 Generating the platform configuration file

In order to have the libraries and configurations in the IDE ready to be used for our board, we need to obtain configuration files, as presented in the Xilinx guide for Kria and Vitis. At the dedicated repository, we can have such configurations, but they required to be built.

`Cmake`, `tcl` and `idn` will become needed in order to build the firmware. As discussed in a thread on Xilinx community forum, `libidn11` is specifically required, but creating a symbolic link from the current, 12 version works.

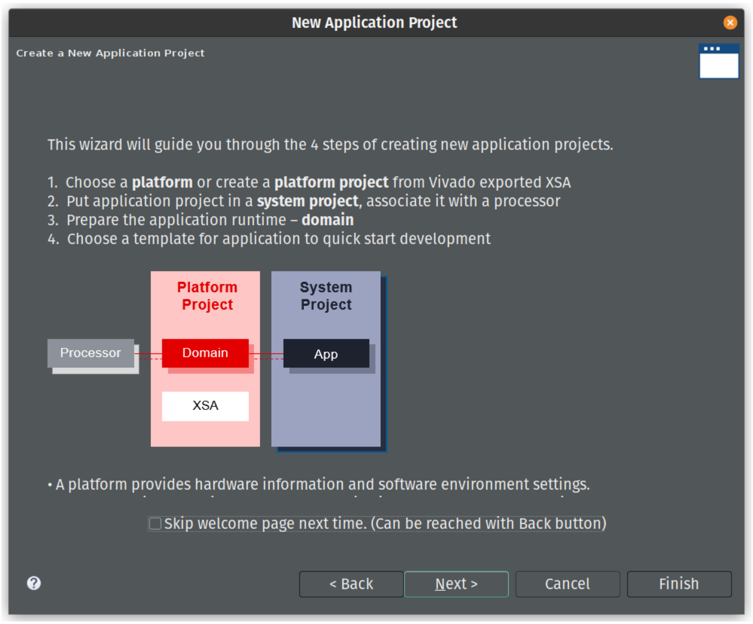
Here are the steps for building this configuration file:

```
sudo apt-get update
sudo apt-get install cmake tcl libidn11-dev \
libidn-dev libidn12 idn
sudo ln -s /usr/lib/x86_64-linux-gnu/libidn.so.12 \
/usr/lib/x86_64-linux-gnu/libidn.so.11

cd ~/Xilinx
git clone --recursive \
https://github.com/Xilinx/kria-vitis-platforms.git
cd kria-vitis-platforms/k26/platforms
export XILINX_VIVADO=/home/$USER/Xilinx/Vivado/2022.2/
export XILINX_VITIS=/home/$USER/Xilinx/Vitis/2022.2/
make platform PLATFORM=k26_base_starter_kit
```

8.2 Setting up a new IDE project for the Kria board

With the platform configuration files available, we can now use the IDE to generate a new project for our board. The whole process will be described with screen captures and captions.


New Application Project

Create a New Application Project

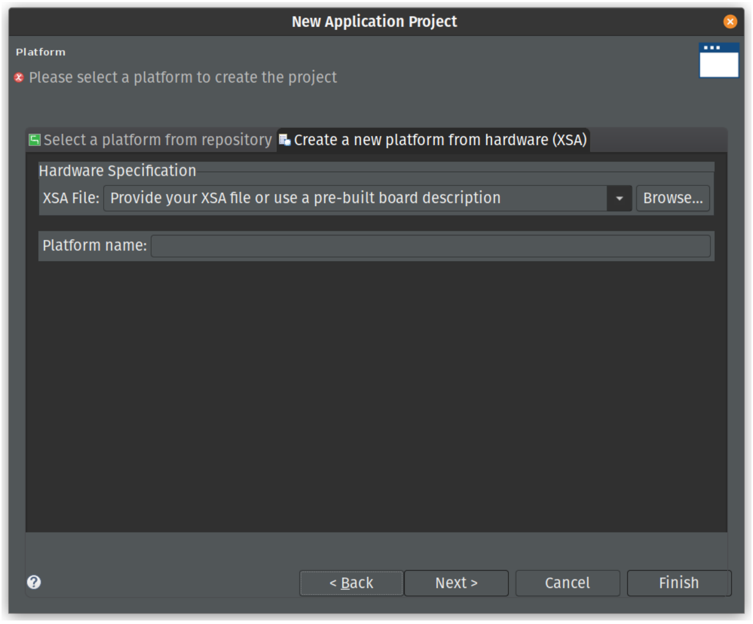
This wizard will guide you through the 4 steps of creating new application projects.

1. Choose a **platform** or create a **platform project** from Vivado exported XSA
2. Put application project in a **system project**, associate it with a processor
3. Prepare the application runtime – **domain**
4. Choose a template for application to quick start development

• A platform provides hardware information and software environment settings.

Skip welcome page next time. (Can be reached with Back button)

Figure 2: We are starting with creating a "New Application Project". You should be greeted with this wizard window. Next.


New Application Project

Platform

Please select a platform to create the project

Select a platform from repository
Create a new platform from hardware (XSA)

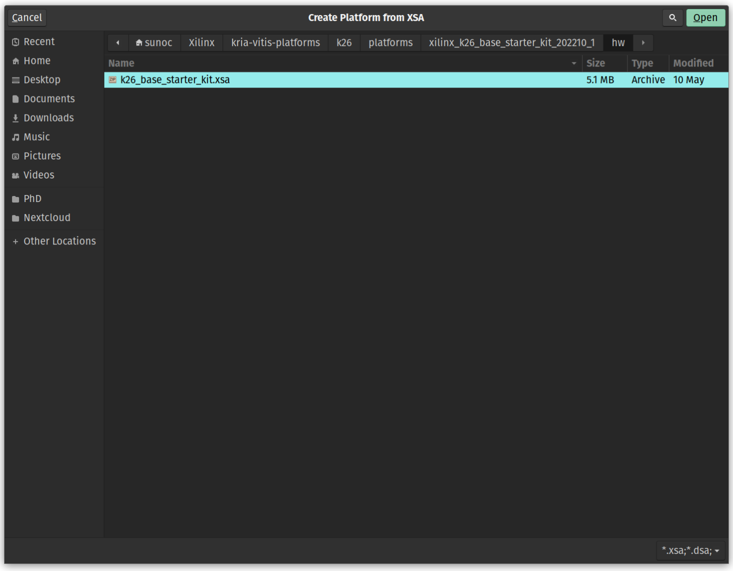
Hardware Specification

XSA File: Provide your XSA file or use a pre-built board description

Platform name:

< Back
Next >
Cancel
Finish

Figure 3: For the platform, we need to get our build Kria configuration. In the "Create a new platform..." tab, click the "Browse..." button.

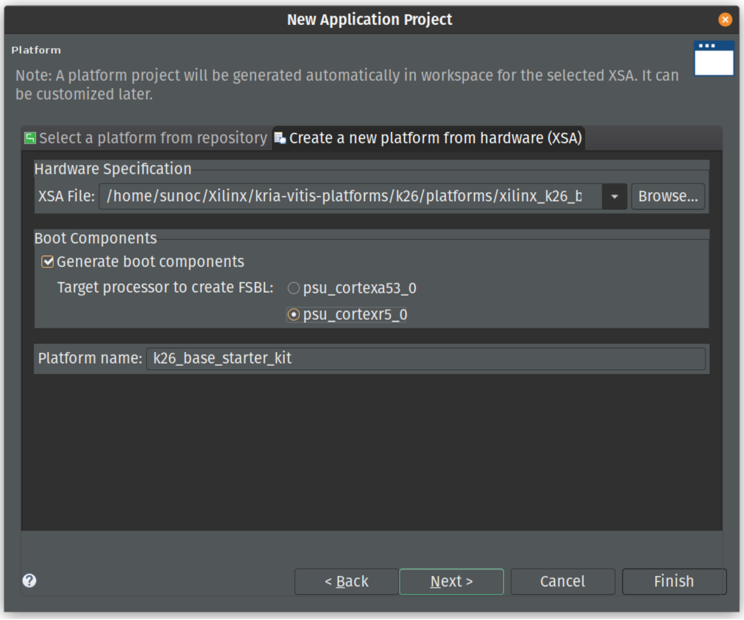


sunoc / Xilinx / kria-vitis-platforms / k26 / platforms / xilinx_k26_base_starter_kit_202210_1 / hw

Name	Size	Type	Modified
k26_base_starter_kit.xsa	5.1 MB	Archive	10 May

.xsa;.dsa;

Figure 4: In the file explorer, we should navigate in the "k26" directory, where the configuration file was build. From here we are looking for a ".xsa" file, located in a "hw" directory, as visible.


New Application Project
Platform
Select a platform from repository
Create a new platform from hardware (XSA)
Browse...
< Back
Next >
Cancel
Finish

Note: A platform project will be generated automatically in workspace for the selected XSA. It can be customized later.

Hardware Specification

XSA File: /home/sunoc/Xilinx/kria-vitis-platforms/k26/platforms/xilinx_k26_b

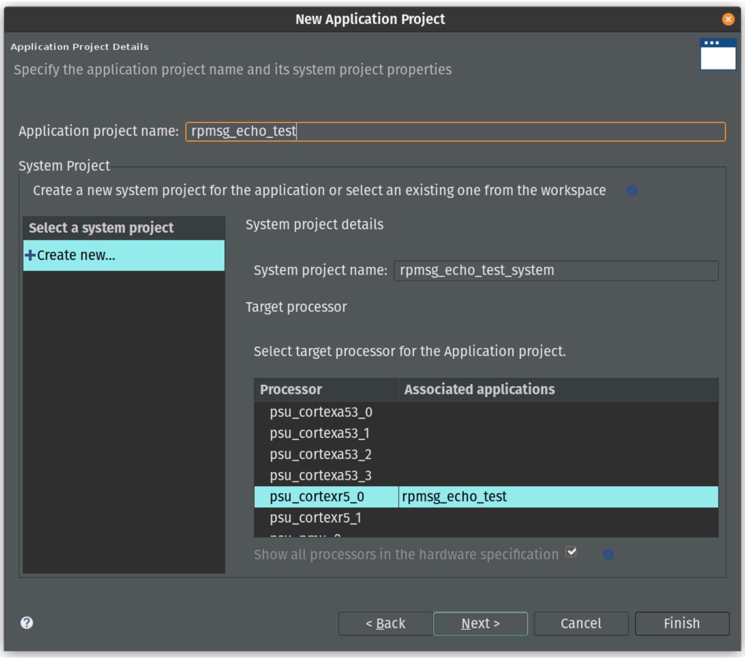
Boot Components

[✓] Generate boot components

Target processor to create FSBL: psu_cortexa53_0 [✓ psu_cortexr5_0]

Platform name: k26_base_starter_kit

Figure 5: With the configuration file loaded, we can now select a name for our platform, but most importantly, we have to select the "psu_cortex5_0" core as a target. The other, Cortex 53 is the APU running Linux.



New Application Project

Application Project Details

Specify the application project name and its system project properties

Application project name: rpmsg_echo_test

System Project

Create a new system project for the application or select an existing one from the workspace

Select a system project

+Create new...

System project details

System project name: rpmsg_echo_test_system

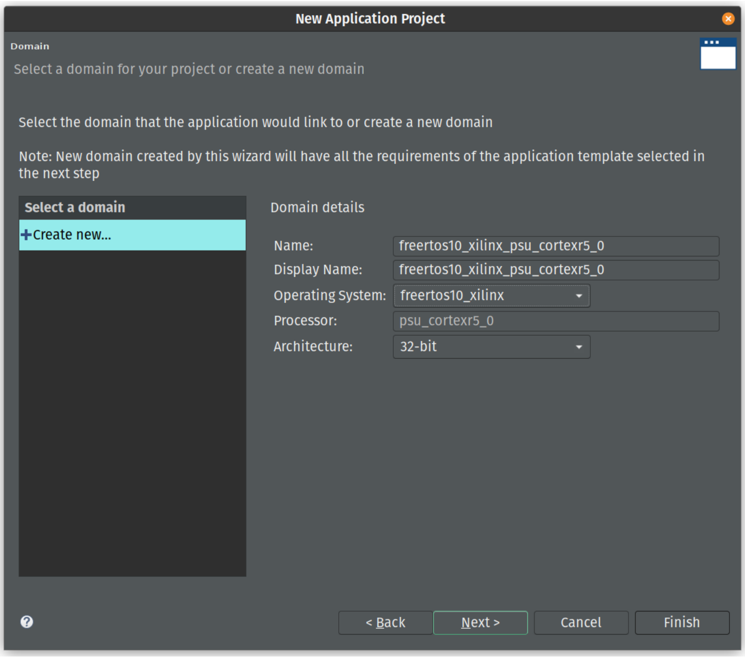
Target processor

Select target processor for the Application project.

Processor	Associated applications
psu_cortexa53_0	
psu_cortexa53_1	
psu_cortexa53_2	
psu_cortexa53_3	
psu_cortexr5_0	rpmsg_echo_test
psu_cortexr5_1	

Show all processors in the hardware specification ✓

Figure 6: In this next window, we can give a name to our firmware project. It is also critical here to select the core we want to build for. Once again, we want to use the "psu_cortex5_0".



New Application Project

Domain

Select a domain for your project or create a new domain

Select the domain that the application would link to or create a new domain

Note: New domain created by this wizard will have all the requirements of the application template selected in the next step

Select a domain

+Create new...

Domain details

Name:	freertos10_xilinx_psu_cortexr5_0
Display Name:	freertos10_xilinx_psu_cortexr5_0
Operating System:	freertos10_xilinx
Processor:	psu_cortexr5_0
Architecture:	32-bit

Figure 7: Here, we want to select "freertos10_xilinx" as our Operating System. The rest can remain unchanged.

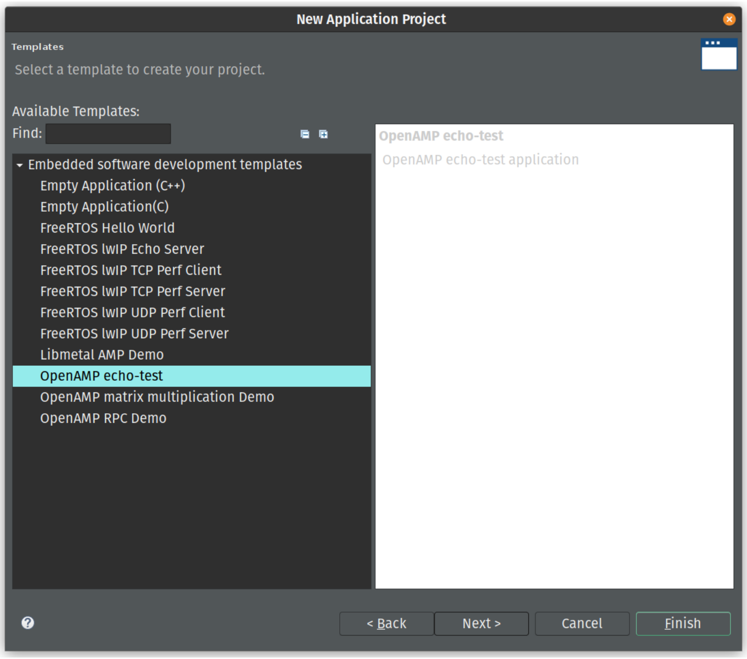

New Application Project
Templates
Select a template to create your project.
Available Templates:
Find:
Embedded software development templates
Empty Application (C++)
Empty Application(C)
FreeRTOS Hello World
FreeRTOS lwIP Echo Server
FreeRTOS lwIP TCP Perf Client
FreeRTOS lwIP TCP Perf Server
FreeRTOS lwIP UDP Perf Client
FreeRTOS lwIP UDP Perf Server
Libmetal AMP Demo
OpenAMP echo-test
OpenAMP matrix multiplication Demo
OpenAMP RPC Demo
OpenAMP echo-test
OpenAMP echo-test application
< Back
Next >
Cancel
Finish

Figure 8: Finally, we can select the demonstration template we are going to use; here we go with "OpenAMP echo-test" since we want to have some simple try of the RPMsg system. Finish.

In the Xilinx documentation, it is made mention of the addresses setting that should be checked in the `lscript.ld` file. These valued look different from what could be set in the DTO for the Linux side, but they appear to work for the example we are running:

```
psu_ddr_S_AXI_BASEADDR            0x3ed00000
psu_ocm_ram_1_S_AXI_BASEADDR      0xfffc0000
psu_r5_tcm_ram_0_S_AXI_BASEADDR   0x00000000
psu_r5_tcm_ram_1_S_AXI_BASEADDR   0x00020000
```

9 Including the micro-ROS static library in a project

Now we have a Vitis demonstration project available and the `libmicroros` static library available, we can combine both by including this library into our Kria project.

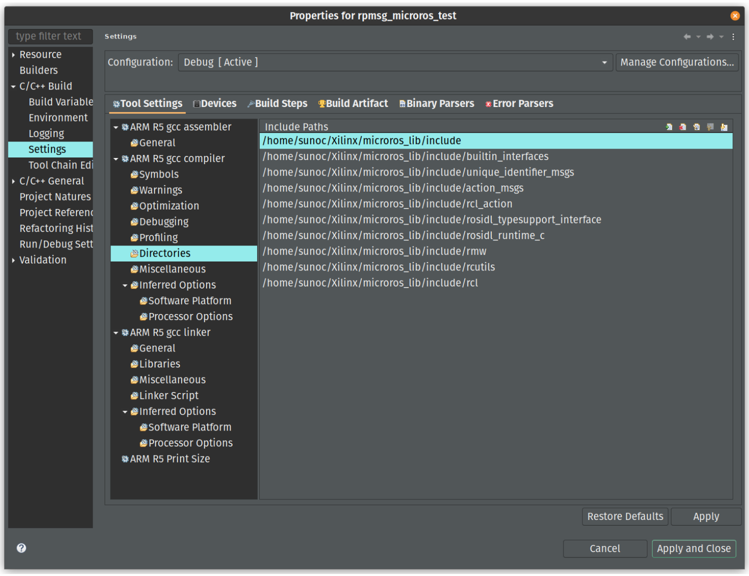
On the host machine running the IDE, we can download the static library and the include files from the build Docker. Here, we assume your Vitis IDE workspace sits in you home directory.

```
mkdir /home/$USER/workspace/microros_lib

docker cp ros_build:/microros_ws/firmware/build/\
  libmicroros.a /home/$USER/workspace/microros_lib/

docker cp ros_build:/microros_ws/firmware/build/include \
/home/$USER/workspace/microros_lib/
```

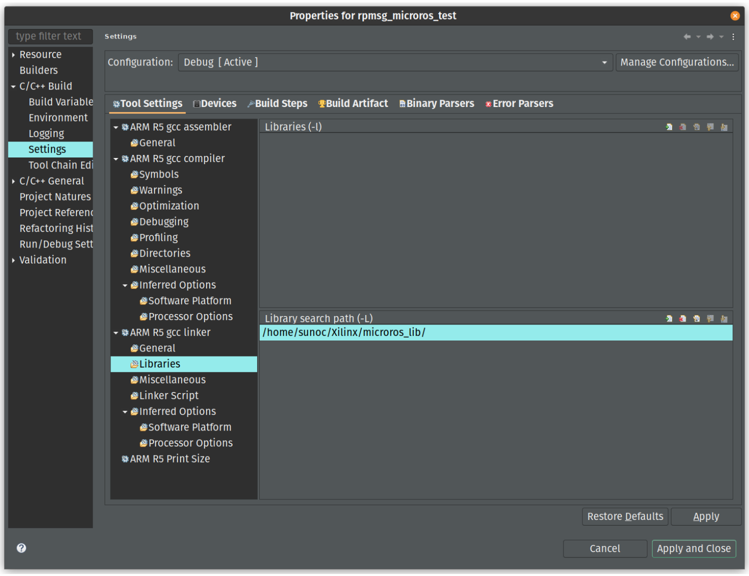
Many parameters are available to be set up in the IDE for the compilation toolchain, but here is a setup that worked to have the IDE to recognize the include files and to be able to use them for compiling the firmware.



Include Paths

```
/home/sunoc/Xilinx/microros_lib/include
/home/sunoc/Xilinx/microros_lib/include/builtin_interfaces
/home/sunoc/Xilinx/microros_lib/include/unique_identifier_msgs
/home/sunoc/Xilinx/microros_lib/include/action_msgs
/home/sunoc/Xilinx/microros_lib/include/rcl_action
/home/sunoc/Xilinx/microros_lib/include/rosidl_typesupport_interface
/home/sunoc/Xilinx/microros_lib/include/rosidl_runtime_c
/home/sunoc/Xilinx/microros_lib/include/rmw
/home/sunoc/Xilinx/microros_lib/include/rcutils
/home/sunoc/Xilinx/microros_lib/include/rcl
```

Figure 9: Firstly, in the "C/C++ Build" settings of your firmware project, under the "Settings" menu, you should find the gcc compiler "Directories". In here you should add the "include" directory of your library. Be careful however, if your include files are in a second layer of directory (as it is the case for libmicroros) you will need to include each sub-directory individually.



Libraries (-l)

Library search path (-L)

```
/home/sunoc/Xilinx/microros_lib/
```

Figure 10: Secondly, in the gcc linkers "Libraries", you can add the top level directory of your library. In our case, it is the directory that contains both the "include" directory added earlier, and also the "libmicroros.a" file.

With both of these setup in your project, you should be able to include the following micro-ROS libraries into your project:

```
#include <rcl/rcl.h>
#include <rcl/error_handling.h>
#include <rclc/rclc.h>
#include <rclc/executor.h>
```

The details for the inclusions and the use-case of the library will depend on the implementation of the firmware itself. But in general, as the firmware is successfully built and an `.elf` file is available, one can upload said firmware to the Kria board (or any remote server accessible through SSH, for that matter) with the following command. Note that in that case, we are retrieving the binary for a project named "rpmsg_pingpong_microros_lib". You own path will vary depending on how your project was named in the first place.

```
scp /home/sunoc/workspace/rpmsg_pingpong_microros_lib/\
  Debug/rpmsg_pingpong_microros_lib.elf  kria:/home/ubuntu/
```

9.1 WIP:: Stream Buffer system enabled

This is a subpart in the general configuration in the project, but this point in particular created so much pain I needed to include in early in this guide for not to forget about it and how to do it.

Indeed, two settings need to be enabled in order to be able to call functions such as `xMessageBufferCreate`:

- Enableling Stream Buffer in the Vitis IDE setting: as show in the figure below, it is needed to tick the settings in the
- Enableling callbacks for the buffers: this is a setting that must be changed in your header file in the project. If you need to use a callback function such as `xMessageBufferCreateWithCallback`, you must include `#define configUSE_SB_COMPLETED_CALLBACK 1` on the top of you header file, before the `#include "FreeRTOS.h"` to override the setting.

TODO ADD THE FIGURE FOR THE STREAM BUFFER SETTING HERE !!!!

10 Using micro-ROS in a Kria demonstration project

Beyond the inclusion of the library itself, actually using the micro-ROS system within an external project require more than just importing the needed functions.

Indeed, if you would be just adding the various function for sending messages to the general ROS2 network, you would face issues with three key aspects, namely:

- Time-related functions, in particular `clock_gettime`.
- Memory allocation functions.
- Communication function.

All three points requires some custom functions in order for the library to know what to do with these types of function it is trying to use. In our case, I will present the implementation of these functions for the Kria board.

10.1 Custom time-related functions

The end result for these implementation are visible in the appendix D.

As micro-ROS can be used on a variety of board, it does not understand by itself what time functions are meant to be used, so some API-style function are being used in the library and it is then needed for the person using a new board to implement these function inner working using the board own time-related function calls.

In particular for this part, the `clock_gettime` function is key, and could simply be implemented with some FreeRTOS time functions. Again, please refer to the appendix D for details.

10.2 WIP:: Custom memory allocator functions

Similarly, it is required to re-implement some form of memory allocating functions in order for the library to be able to work with such functionalities in a formalized way.

As for now, the current version of the allocator function can be seen in the appendix E, but the current setup is not working properly yet and some further test and modification will be needed.

10.3 WIP:: Custom transport layer functions

This part is the key translation layer that needs to happen in order for the DDS system from the micro-ROS library to be using the communication channel we want it to.

11 Loading a new firmware from Linux

Having now the latest version of our firmware (with or without the included micro-ROS library) loaded onto our Kria's Linux, we want to load and run it on the Cortex micro-controller side.

Here are the instructions used to upload and start the firmware on the R5F from the Linux. In this sequence, we are entering a root shell with `sudo -s`, but this can also be archived by putting the commands in a script to be executed with `sudo`. It is also important to note that the `echo_test` part is specific for the RPMsg base demonstration firmware. It is not to be used for other firmware (including the one involving micro-ROS).

```
sudo -s
cp image_echo_test /lib/firmware
echo image_echo_test  > /sys/class/remoteproc/\
remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
echo_test
echo stop > /sys/class/remoteproc/remoteproc0/state
```

This was put in a script that can be run as a admin. It is to be noted that as for the current testing, it is more convenient to run the firmware and to stop it manually afterward. Thus the script would look like this:

```
#!/bin/sh
rm -f /lib/firmware/firmware.elf
mv firmware.elf /lib/firmware
echo firmware.elf > /sys/class/remoteproc/remoteproc0/firmware
echo start > /sys/class/remoteproc/remoteproc0/state
```

And the sequence to run to start, test, stop and restart the script would be as follow:

```
  sudo -s
  bash firmware_script.sh
  echo_test
  echo stop > /sys/class/remoteproc/remoteproc0/state
  echo start > /sys/class/remoteproc/remoteproc0/state
```

12 ROS2

As an Ubuntu distribution is installed on the board, the installation of ROS2 can be done in a standard way, using the repository.

An official documentation is provided with ROS2 themselves with a step-by-step guide on how to install ROS2 on a Ubuntu system[6]. We will follow this guide here.

Firstly, we need to update the locals, enable the universe Ubuntu repository, get the key and add the repository for ROS2. This can be done as follow:

```
locale  # check for UTF-8
sudo apt update && sudo apt install -y locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  # verify settings
sudo apt install -y software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install -y curl wget
wget https://raw.githubusercontent.com/ros/rosdistro/\
master/ros.key
sudo mv ros.key /usr/share/keyrings/ros-archive-keyring.gpg
```

This is a one-liner to add the ROS2 repository:

```
echo "deb [arch=$(dpkg --print-architecture) \
signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] \
http://packages.ros.org/ros2/ubuntu $(. \
/etc/os-release && echo $UBUNTU_CODENAME) main" | \
sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

It is then possible to install ROS2[7] as follow:

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y ros-humble-desktop \
ros-humble-ros-base \
python3-argcomplete \
ros-dev-tools
```

Once installed, it is possible to test the system with a provided example. You need to open two terminals and log wish SSH onto the board, then running respectively:

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_cpp talker
```

```
source /opt/ros/humble/setup.bash
ros2 run demo_nodes_py listener
```

[6]The curl command from the guide does not work through the school proxy, but the command wget used instead does work. The key is then moved to the correct spot with mv

[7]This command installs a complete "desktop" version of ROS2. If space is a constraint, different, less complete packages can be install. Please refer to the official documentation about it.

You should be able to see the first terminal sending "Hello world" messages, and the second one receiving then.

13 ROS2 in a container

As it is used to test and build micro-ROS configurations, running ROS2 in a Docker container is a great way to have a reproducible configuration of you system. This part of the guide will present how to install Docker on the Kria board and then how to use it to deploy the latest version of ROS2.

13.1 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 on the figure 11 below:

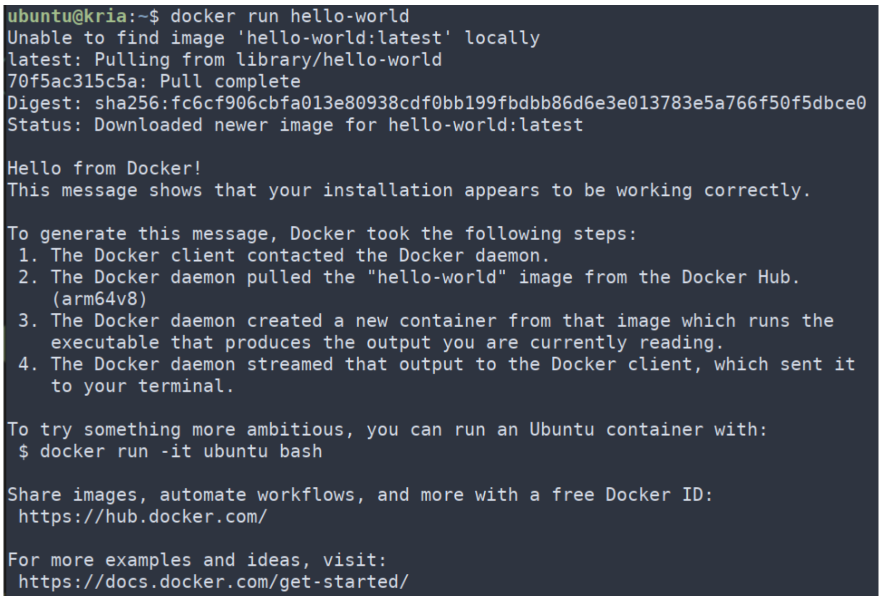
```
sudo apt-get update
sudo apt-get install ca-certificates curl sudo
gnupg install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | \
sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

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 11: The return of a successful run of the “hello world” test Docker container

13.2 Calling a ROS2 container

The following commands will pull a ROS container, version `humble`, and name it `ros_build`. A key part for having access to the interfaces (serial) is the mapping of whole the `/dev` of the host machine to the internal `/dev`. [8] With the second command, we can execute `bash`, as a way to open a terminal to the “inside” of the container.

```
sudo docker run -d --name ros_agent -it --net=host -v \
     /dev:/dev --privileged ros:iron
sudo docker exec -it ros_agent bash
```

From there, it becomes possible to simply use ROS2 as you would for a bare-metal install, and as presented in the section 12. The following lines show the whole procedure on how to enable a micro-ROS agent that will listen and talk using UART on the `/dev/ttyUSB1` port.

[8]Note that this is an example and this situation can become a security issue. It would be a better practice in a production environment to map only the devices that are actually in use.

```
  source /opt/ros/$ROS_DISTRO/setup.bash

  # Create a workspace and download the micro-ROS tools
  mkdir microros_ws
  cd microros_ws
  git clone -b $ROS_DISTRO https://github.com/micro-ROS/\
  micro_ros_setup.git src/micro_ros_setup

  # Update dependencies using rosdep
  sudo apt update && rosdep update
  rosdep install --from-paths src --ignore-src -y

  # Install pip
  sudo apt-get install python3-pip

  # Build micro-ROS tools and source them
  colcon build
  source install/local_setup.bash
```

```
# Download micro-ROS-Agent packages
source install/local_setup.bash
ros2 run micro_ros_setup create_agent_ws.sh

# Build step
ros2 run micro_ros_setup build_agent.sh
source install/local_setup.bash

# Run a micro-ROS agent
ros2 run micro_ros_agent micro_ros_agent serial \
--dev /dev/ttyUSB1

q
```

Samewise, from a different shell, it is possible to run a demonstration ping-pong topic communication. Note that you need to be careful to have you shell in the "correct" space: these command need to be run inside the container in which the previous setup were install, not on the host running the container system.

```
  source /opt/ros/$ROS_DISTRO/setup.bash

  # Subscribe to micro-ROS ping topic
  ros2 topic echo /microROS/ping
```

14 micro-ROS custom agent

https://github.com/micro-ROS/micro_ros_setup/issues/383
https://github.com/micro-ROS/micro_ros_setup/issues/591
https://micro.ros.org/docs/tutorials/advanced/create_custom_transports/
https://github.com/eProsima/Micro-XRCE-DDS-Agent/blob/develop/src/cpp/transport/custom/CustomAgent.cpp
/microros_ws/build/micro_ros_agent/agent/src/xrceagent/src/cpp/transport/custom/CustomAgent.cpp

A DTO patch file

```
diff -u --label /ssh\:kria\:/home/ubuntu/system_original.dts --label /ssh\:kria\:/home/ubuntu/system.dts /tmp/tramp.KJbEbz.dts /tmp/tramp.zHBG7v.dts
--- /ssh:kria:/home/ubuntu/system_original.dts
+++ /ssh:kria:/home/ubuntu/system.dts
@@ -138,7 +138,7 @@

         firmware {

-                zynqmp-firmware {
+                zynqmp_firmware: zynqmp-firmware {
                         compatible = "xlnx,zynqmp-firmware";
                         #power-domain-cells = <0x01>;
                         method = "smc";
@@ -719,7 +719,7 @@
                         phandle = <0x44>;
                 };

-                interrupt-controller@f9010000 {
+                gic: interrupt-controller@f9010000 {
                         compatible = "arm,gic-400";
                         #interrupt-cells = <0x03>;
                         reg = <0x00 0xf9010000 0x00 0x10000 0x00 0xf9020000 0x00 0x20000 0x00 0xf9040000 0x00 0x20000 0x00 0xf9060000 0x00 0x20000>;
@@ -1536,7 +1536,7 @@
                         pinctrl-names = "default";
                         u-boot,dm-pre-reloc;
                         compatible = "xlnx,zynqmp-uart\0cdns,uart-r1p12";
-                        status = "okay";
+                        status = "disabled";
                         interrupt-parent = <0x04>;
                         interrupts = <0x00 0x16 0x04>;
                         reg = <0x00 0xff010000 0x00 0x1000>;
@@ -1909,6 +1909,84 @@
                 pwms = <0x1b 0x02 0x9c40 0x00>;
         };

+       reserved-memory {
+           #address-cells = <2>;
+           #size-cells = <2>;
+           ranges;
+           rpu0vdev0vring0: rpu0vdev0vring0@3ed40000 {
+               no-map;
+               reg = <0x0 0x3ed40000 0x0 0x4000>;
+           };
+           rpu0vdev0vring1: rpu0vdev0vring1@3ed44000 {
+               no-map;
+               reg = <0x0 0x3ed44000 0x0 0x4000>;
+           };
+           rpu0vdev0buffer: rpu0vdev0buffer@3ed48000 {
+               no-map;
+               reg = <0x0 0x3ed48000 0x0 0x100000>;
+           };
+           rproc_0_reserved: rproc_0_reserved@3ed00000 {
+               no-map;
+               reg = <0x0 0x3ed00000 0x0 0x40000>;
+           };
+       };
+       tcm_0a: tcm_0a@ffe00000 {
+           no-map;
+           reg = <0x0 0xffe00000 0x0 0x10000>;
+           status = "okay";
+           compatible = "mmio-sram";
+           power-domain = <&zynqmp_firmware 15>;
+       };
+       tcm_0b: tcm_0b@ffe20000 {
+           no-map;
+           reg = <0x0 0xffe20000 0x0 0x10000>;
+           status = "okay";
+           compatible = "mmio-sram";
+           power-domain = <&zynqmp_firmware 16>;
+       };
+       rf5ss@ff9a0000 {
+           compatible = "xlnx,zynqmp-r5-remoteproc";
+           xlnx,cluster-mode = <1>;
+           ranges;
+           reg = <0x0 0xFF9A0000 0x0 0x10000>;
+           #address-cells = <0x2>;
+           #size-cells = <0x2>;
+           r5f_0 {
+               compatible = "xilinx,r5f";
+               #address-cells = <2>;
+               #size-cells = <2>;
+               ranges;
+               sram = <&tcm_0a &tcm_0b>;
+               memory-region = <&rproc_0_reserved>, <&rpu0vdev0buffer>,
↪   <&rpu0vdev0vring0>, <&rpu0vdev0vring1>;
+               power-domain = <&zynqmp_firmware 7>;
+               mboxes = <&ipi_mailbox_rpu0 0>, <&ipi_mailbox_rpu0 1>;
+               mbox-names = "tx", "rx";
+           };
+       };
+       zynqmp_ipi1 {
+           compatible = "xlnx,zynqmp-ipi-mailbox";
+           interrupt-parent = <&gic>;
+           interrupts = <0 29 4>;
+           xlnx,ipi-id = <7>;
+           #address-cells = <1>;
+           #size-cells = <1>;
+           ranges;
+           /* APU<->RPU0 IPI mailbox controller */
+           ipi_mailbox_rpu0: mailbox@ff990600 {
+               reg = <0xff990600 0x20>,
+                     <0xff990620 0x20>,
+                     <0xff9900c0 0x20>,
+                     <0xff9900e0 0x20>;
+               reg-names = "local_request_region",
+                               "local_response_region",
+                               "remote_request_region",
+                               "remote_response_region";
+                       #mbox-cells = <1>;
+                       xlnx,ipi-id = <1>;
+                   };
+               };
+
+
            __symbols__ {
                    cpu0 = "/cpus/cpu@0";
                    cpu1 = "/cpus/cpu@1";

Diff finished.  Wed May 24 10:15:49 2023
```

B Custom toolchain CMake settings

```
set(CMAKE_SYSTEM_NAME Generic)
set(CMAKE_CROSSCOMPILING 1)
set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)
set(CMAKE_INSTALL_LIBDIR /usr/)
set(PLATFORM_NAME "LwIP")


set(ARCH_CPU_FLAGS "-mcpu=cortex-r5 -mthumb -mfpu=vfpv3-d16 -mfloat-abi=hard -DARMR5 -O0
↪  -Wall -fdata-sections -ffunction-sections -fno-tree-loop-distribute-patterns
↪  -Wno-unused-parameter -Wno-unused-value -Wno-unused-variable -Wno-unused-function
↪  -Wno-unused-but-set-variable" CACHE STRING "" FORCE)
set(ARCH_OPT_FLAGS "")

set(CMAKE_C_COMPILER arm-none-eabi-gcc)
set(CMAKE_CXX_COMPILER arm-none-eabi-g++)

set(CMAKE_C_FLAGS_INIT "-std=c11 ${ARCH_CPU_FLAGS} ${ARCH_OPT_FLAGS} -DCLOCK_MONOTONIC=0"
↪  CACHE STRING "" FORCE)
set(CMAKE_CXX_FLAGS_INIT "-std=c++14 ${ARCH_CPU_FLAGS} ${ARCH_OPT_FLAGS}
↪  -DCLOCK_MONOTONIC=0" CACHE STRING "" FORCE)



set(__BIG_ENDIAN__ 0)
```

C Custom Colcon meta settings

```json
{
    "names": {
        "tracetools": {
            "cmake-args": [
                "-DTRACETOOLS_DISABLED=ON",
                "-DTRACETOOLS_STATUS_CHECKING_TOOL=OFF"
            ]
        },
        "rosidl_typesupport": {
            "cmake-args": [
               "-DROSIDL_TYPESUPPORT_SINGLE_TYPESUPPORT=ON"
            ]
        },
        "rcl": {
            "cmake-args": [
                "-DBUILD_TESTING=OFF",
                "-DRCL_COMMAND_LINE_ENABLED=OFF",
                "-DRCL_LOGGING_ENABLED=OFF"
            ]
        },
        "rcutils": {
            "cmake-args": [
                "-DENABLE_TESTING=OFF",
                "-DRCUTILS_NO_FILESYSTEM=ON",
                "-DRCUTILS_NO_THREAD_SUPPORT=ON",
                "-DRCUTILS_NO_64_ATOMIC=ON",
                "-DRCUTILS_AVOID_DYNAMIC_ALLOCATION=ON"
            ]
        },
        "microxrcedds_client": {
            "cmake-args": [
                "-DUCLIENT_PIC=OFF",
                "-DUCLIENT_PROFILE_UDP=OFF",
                "-DUCLIENT_PROFILE_TCP=OFF",
                "-DUCLIENT_PROFILE_DISCOVERY=OFF",
                "-DUCLIENT_PROFILE_SERIAL=OFF",
                "-UCLIENT_PROFILE_STREAM_FRAMING=ON",
                "-DUCLIENT_PROFILE_CUSTOM_TRANSPORT=ON"
            ]
        },
        "rmw_microxrcedds": {
            "cmake-args": [
                "-DRMW_UXRCE_MAX_NODES=1",
                "-DRMW_UXRCE_MAX_PUBLISHERS=5",
                "-DRMW_UXRCE_MAX_SUBSCRIPTIONS=5",
                "-DRMW_UXRCE_MAX_SERVICES=1",
                "-DRMW_UXRCE_MAX_CLIENTS=1",
                "-DRMW_UXRCE_MAX_HISTORY=4",
                "-DRMW_UXRCE_TRANSPORT=custom"
            ]
        }
    }
}
```

D Firmware time functions

D.1 Main function

```c
#include "microros.h"


int _gettimeofday( struct timeval *tv, void *tzvp )
{
        XTime t = 0;
    XTime_GetTime(&t);  //get uptime in nanoseconds
    tv->tv_sec = t / 1000000000;  // convert to seconds
    tv->tv_usec = ( t % 1000000000 ) / 1000;  // get remaining microseconds
    return 0;  // return non-zero for error
} // end _gettimeofday()


void UTILS_NanosecondsToTimespec( int64_t llSource,
                                  struct timespec * const pxDestination )
{
    long lCarrySec = 0;

    /* Convert to timespec. */
    pxDestination->tv_sec = ( time_t ) ( llSource / NANOSECONDS_PER_SECOND );
    pxDestination->tv_nsec = ( long ) ( llSource % NANOSECONDS_PER_SECOND );

    /* Subtract from tv_sec if tv_nsec < 0. */
    if( pxDestination->tv_nsec < 0L )
    {
        /* Compute the number of seconds to carry. */
        lCarrySec = ( pxDestination->tv_nsec / ( long ) NANOSECONDS_PER_SECOND ) + 1L;

        pxDestination->tv_sec -= ( time_t ) ( lCarrySec );
        pxDestination->tv_nsec += lCarrySec * ( long ) NANOSECONDS_PER_SECOND;
    }
}

int clock_gettime( clockid_t clock_id,
                   struct timespec * tp )
{
        TimeOut_t xCurrentTime = { 0 };

    /* Intermediate variable used to convert TimeOut_t to struct timespec.
     * Also used to detect overflow issues. It must be unsigned because the
     * behavior of signed integer overflow is undefined. */
    uint64_t ullTickCount = 0ULL;

    /* Silence warnings about unused parameters. */
    ( void ) clock_id;

    /* Get the current tick count and overflow count. vTaskSetTimeOutState()
     * is used to get these values because they are both static in tasks.c. */
    vTaskSetTimeOutState( &xCurrentTime );

    /* Adjust the tick count for the number of times a TickType_t has overflowed.
     * portMAX_DELAY should be the maximum value of a TickType_t. */
    ullTickCount = ( uint64_t ) ( xCurrentTime.xOverflowCount ) << ( sizeof( TickType_t )
↪   * 8 );

    /* Add the current tick count. */
    ullTickCount += xCurrentTime.xTimeOnEntering;

    /* Convert ullTickCount to timespec. */
    UTILS_NanosecondsToTimespec( ( int64_t ) ullTickCount * NANOSECONDS_PER_TICK, tp );

    return 0;
}
```

D.2 Header file with definitions

```c
  /**< Microseconds per second. */
  #define MICROSECONDS_PER_SECOND    ( 1000000LL )
  /**< Nanoseconds per second. */
  #define NANOSECONDS_PER_SECOND     ( 1000000000LL )
  /**< Nanoseconds per FreeRTOS tick. */
  #define NANOSECONDS_PER_TICK       ( NANOSECONDS_PER_SECOND / configTICK_RATE_HZ )
```

E Firmware memory allocation functions

E.1 Main function

```c
#include "allocators.h"

//int absoluteUsedMemory = 0;
//int usedMemory = 0;

void * __freertos_allocate(size_t size, void * state){
  (void) state;
  LPRINTF("-- Alloc %d (prev: %d B)\r\n",size, xPortGetFreeHeapSize());
//  absoluteUsedMemory += size;
//  usedMemory += size;

  LPRINTF("Return for the allocate function w parameter size = %d\r\n", size);

  return pvPortMalloc(size);
}

void __freertos_deallocate(void * pointer, void * state){
  (void) state;
  LPRINTF("-- Free 0x%x (prev: %d B)\r\n", pointer, xPortGetFreeHeapSize());
  if (NULL != pointer)
  {
//          LPRINTF("Pointer is not null.\r\n");
//          usedMemory -= getBlockSize(pointer);
//          LPRINTF("usedMemory var updated: %d\r\n", usedMemory);
          vPortFree(pointer);
  }
  else
  {
          LPERROR("Trying to deallocate a null pointed. Doing nothing.\r\n");
  }
}

void * __freertos_reallocate(void * pointer, size_t size, void * state){
  (void) state;
  LPRINTF("-- Realloc 0x%x -> %d (prev: %d B)\r\n", pointer, size,
↪  xPortGetFreeHeapSize());
//  absoluteUsedMemory += size;
//  usedMemory += size;
  if (NULL == pointer)
  {
    return __freertos_allocate(size, state);
  }
  else
  {
//    usedMemory -= getBlockSize(pointer);
//
//    return pvPortRealloc(pointer,size);

          __freertos_deallocate(pointer, state);
          return __freertos_allocate(size, state);

  }
}

void * __freertos_zero_allocate(size_t number_of_elements, size_t size_of_element, void *
↪  state){
  (void) state;
  LPRINTF("-- Calloc %d x %d = %d -> (prev: %d
↪  B)\r\n",number_of_elements,size_of_element, number_of_elements*size_of_element,
↪  xPortGetFreeHeapSize());
//  absoluteUsedMemory += number_of_elements*size_of_element;
//  usedMemory += number_of_elements*size_of_element;

  return pvPortCalloc(number_of_elements,size_of_element);
}
```

E.2 Header file with definitions

```c
#ifndef _ALLOCATORS_H_
#define _ALLOCATORS_H_

#include "microros.h"

extern int absoluteUsedMemory;
extern int usedMemory;


void * __freertos_allocate(size_t size, void * state);
void __freertos_deallocate(void * pointer, void * state);
void * __freertos_reallocate(void * pointer, size_t size, void * state);
void * __freertos_zero_allocate(size_t number_of_elements,
size_t size_of_element, void * state);

#endif // _ALLOCATORS_H_
```