


Setting up and using Xilinx KRIA KV260

南山大学

2023-8-24

Vincent Conus - Source available at [GitLab](#) 



Contents

1	Introduction & motivation	2
2	Boot firmware	2
2.1	Getting the new firmware	2
2.2	Reaching the board recovery tool	2
2.3	Updating the boot firmware	2
3	Installing Linux	2
3.1	Preparing and booting a Ubuntu 22.04 media	3
3.2	Network and admin setups	4
3.2.1	Proxy and DNS	4
3.2.2	root password	4
3.2.3	Static IP address	5
3.2.4	Purging snap	5
3.2.5	Other unused heavy packages	6
4	Enabling remoteproc	6
4.1	Device-Tree Overlay patching	6
5	Building an example RPMMsg real-time firmware	7
5.1	Setting up the IDE	7
5.1.1	Dependencies & installation	8
5.1.2	Platform configuration file generation	8
5.2	Setting up and building a new project for the Kria board	9
6	Building a real-time firmware	13
6.1	Setting up Vitis IDE	13
7	RPMMsg echo_test software	13
8	Building micro-ROS as a static library	13
9	Adding micro-ROS to a firmware project	14
10	Loading a real-time firmware	14
11	Running a ROS2 node	14
11.1	On the host Linux	14
11.2	In a container	14
12	micro-ROS agent	14
A	DTO patch	15
B	Custom toolchain CMake settings	18
C	Custom Colcon meta settings	19
D	Firmware time functions	21
D.1	main	21
D.2	header file	22
E	Firmware memory allocation functions	23
E.1	main	23
E.2	header file	24

1 Introduction & motivation

This guide will present how to setup and use Xilinx's KRIA board, in particular for running ROS on a host Ubuntu system, as well as for deploying micro-ROS as a firmware on the MCU part of this board's chip.

The use of this device in particular is interesting because of the presence of a CPU comprising both a general purpose ARM core, capable of running a Linux distribution, as well as another ARM core, real-time enabled, capable to run a RTOS.

2 Boot firmware

The goal for the Linux side of the deployment is to have the latest LTS version of Ubuntu up and running. In order to be able to boot such a newer version of Linux, the boot image of the board must first 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, or even directly with the following command:

```
1 wget https://www.xilinx.com/member/forms/download/\
2 design-license-xef.html?filename=B00T-k26-starter-kit-20230516185703.bin
```

2.2 Reaching the board recovery tool

Now the firmware .bin 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 a Ethernet cable. This will obviously cut you internet access, so you should be set for that.
- 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.

2.3 Updating the boot firmware

From this "recovery" page, it is possible to upload the .bin file downloaded previously onto the board using the "Recover Image" section at the bottom right of the page.

The board can be re-booted afterwards.

3 Installing Linux

With the boot firmware being up-to-date, we can proceed to install a Linux distribution on our Kria board. The step needed to archive a full installation of Ubuntu 22.04 will be presented in this section.

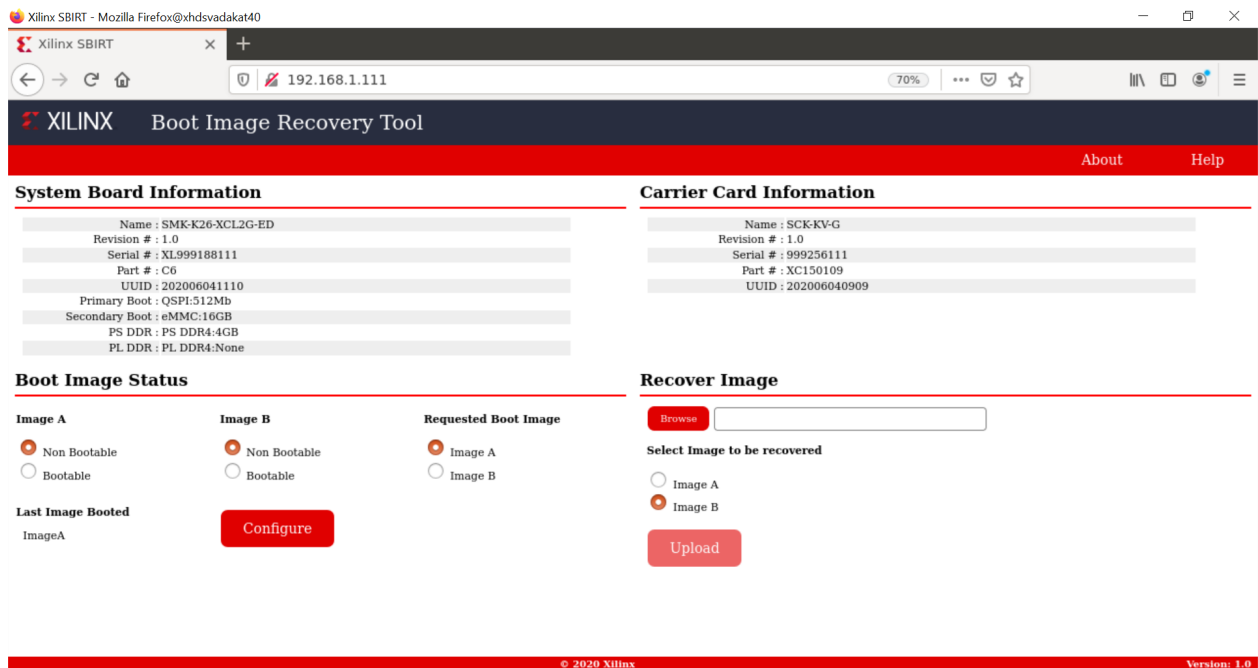


Figure 1: The recovery tool for the board, access from Firefox. We can see board information at the center, and the tools to upload the firmware at the bottom of the page.

3.1 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` 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, and the 115200 bitrate is the default value for the board):

```
1 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, with the first command to be run on the board and the second one on the host PC, both without the first placeholder hostnames):

```
1 kria# ip addr
```

```
3 host# ssh ubuntu@192.168.4.11
```

3.2 Network and admin setups

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

3.2.1 Proxy and DNS

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.

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. The resolver can then be restarted.

```
1 sudo nano /etc/resolv.conf
2
3 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 (this is specific to Nanzan University IT system), then by consolidating the configuration for other types of connections in the `bashrc`:

```
1 export https_proxy="http://<AXIA_username>:\
2     <AXIA_psw>@proxy.ic.nanzan-u.ac.jp:8080"
3
4 echo "export http_proxy=\"\"$https_proxy\"\" >> ~/.bashrc \
5     echo "export https_proxy=\"\"$https_proxy\"\" >> ~/.bashrc \
6     echo "export ftp_proxy=\"\"$https_proxy\"\" >> ~/.bashrc \
7     echo "export no_proxy=\"localhost, 127.0.0.1,::1\"\" \
8     >> ~/.bashrc
```

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

3.2.2 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:

```
1 sudo visudo
2
3 %sudo    ALL=(ALL:ALL) NOPASSWD: ALL
```

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:

```
1 sudo apt-get update
2 sudo apt-get dist-upgrade
3 sudo reboot now
```

3.2.3 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:

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

You can then set the wanted IP as follow. Note that a custom DNS was also set in that case.

```
1 network:
2   renderer: NetworkManager
3   version: 2
4   ethernets:
5     eth0:
6       addresses:
7         - 192.168.11.103/24
8       routes:
9         - to: default
10           via: 192.168.11.1
11       nameservers:
12         addresses:
13           - 8.8.8.8
14           - 1.1.1.1
```

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

```
1 sudo netplan apply
```

3.2.4 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 order¹, but the system fan runs sensibly slower without all of this stuff:

```
1 sudo systemctl disable snapd.service
2 sudo systemctl disable snapd.socket
3 sudo systemctl disable snapd.seeded.service
4
5 sudo snap list #show installed package, remove then all:
6 sudo snap remove --purge firefox
7 sudo snap remove --purge gnome-3-38-2004
8 sudo snap remove --purge gnome-42-2204
9 sudo snap remove --purge gtk-common-themes
10 sudo snap remove --purge snapd-desktop-integration
11 sudo snap remove --purge snap-store
```

¹The snap package depends on each other. Thus dependencies cannot be remove before the package(s) that depends on them.

```

12  sudo snap remove --purge bare
13  sudo snap remove --purge core20
14  sudo snap remove --purge core22
15  sudo snap remove --purge snapd
16  sudo snap list # check that everything is uninstalled
17
18  sudo rm -rf /var/cache/snapd/
19  sudo rm -rf ~/snap
20  sudo apt autoremove --purge snapd
21
22  systemctl list-units | grep snapd

```

3.2.5 Other unused heavy packages

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

```

1  sudo apt-get autoremove --purge yaru-theme-icon \
2  fonts-noto-cjk yaru-theme-gtk vim-runtime \
3  ubuntu-wallpapers-jammy humanity-icon-theme
4
5  sudo apt-get autoclean
6  sudo reboot now

```

4 Enabling remoteproc

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 guides²,³ 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.

4.1 Device-Tree Overlay 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.

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

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

```

1  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 but also in this report appendix A.

²A slideshow (JP) from Fixstar employees presents how to use the device tree to enable the communication between the cores.

³A blog post (JP) shows all major steps on how to enable the remoteproc.

```

1 wget https://gitlab.com/sunoc/xilinx-kria-kv260-documentation/-/\
2     blob/b7300116e153f4b5a1542f8804e4646db8030033/src/system.patch
3
4 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 needed⁴:

```

1 sudo nano /etc/default/flash-kernel
2
3 LINUX_KERNEL_CMDLINE="quiet splash cma=512M cpuidle.off=1"
4 LINUX_KERNEL_CMDLINE_DEFAULTS=""
5 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:

```

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

```

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

```

1 ls /sys/class/remoteproc/
2 # remoteproc0

```

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

5 Building an example RPMsg real-time firmware

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.

5.1 Setting up the IDE

Xilinx's Vitis IDE is the recommended tool 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).

However, this large tool-set allows for a convenient development environment, in particular in our case where some FreeRTOS system, with many dependencies is to be build.

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.bin⁵.

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. However, even with this more unstable status, no major problems were encountered with this tool during the development stages.

This guide will present a setup procedure 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.

⁴The 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.

⁵The name of the installer binary file might change as a new version of the IDE is release every year or so.

5.1.1 Dependencies & installation

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

```
1 sudo apt-get -y update
2
3 sudo apt-get -y install libncurses-dev \
4     ncurses-term \
5     ncurses-base \
6     ncurses-bin \
7     libncurses5 \
8     libtinfo5 \
9     libncurses5-dev \
10    libncursesw5-dev
```

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

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

If it is not possible to run the previous command, make the file executable with the `chmod` command:

```
1 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.

WARNING: This whole procedure can take up to multiple hours to complete and is prone to failures (regarding missing dependencies, typically), so your schedule should be arranged accordingly.

5.1.2 Platform configuration file generation

In order to have the libraries and configurations in the IDE ready to be used for our board, we need to obtain some configuration files that are specific for the Kria KV260, as presented in the Xilinx guide for Kria and Vitis.

A Xilinx dedicated repository is available for us to download such configurations, but they required to be built.

As for the dependencies, Cmake, tcl and idn will become needed in order to build the firmware. Regarding idn, some version issue can happen, but as discussed in a thread on Xilinx's forum, if libidn11 is specifically required but not available (it is the case for Ubuntu 22.04), creating a symbolic link from the current, 12 version works as a workaround.

Here are the steps for installing the dependencies and building this configuration file:

```
1 sudo apt-get update
2 sudo apt-get install cmake tcl libidn11-dev \
3     libidn-dev libidn2 idn
4 sudo ln -s /usr/lib/x86_64-linux-gnu/libidn.so.12 \
5     /usr/lib/x86_64-linux-gnu/libidn.so.11
6
7 cd ~/Xilinx
8 git clone --recursive \
9     https://github.com/Xilinx/kria-vitis-platforms.git
10 cd kria-vitis-platforms/k26/platforms
11 export XILINX_VIVADO=/home/$USER/Xilinx/Vivado/2022.2/
12 export XILINX_VITIS=/home/$USER/Xilinx/Vitis/2022.2/
13 make platform PLATFORM=k26_base_starter_kit
```

5.2 Setting up and building a new 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.

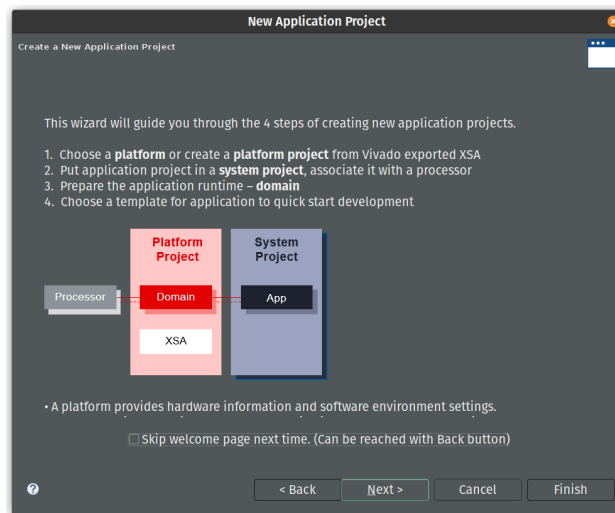


Figure 2: We are starting with creating a "New Application Project" You should be greeted with this wizard window. Next.

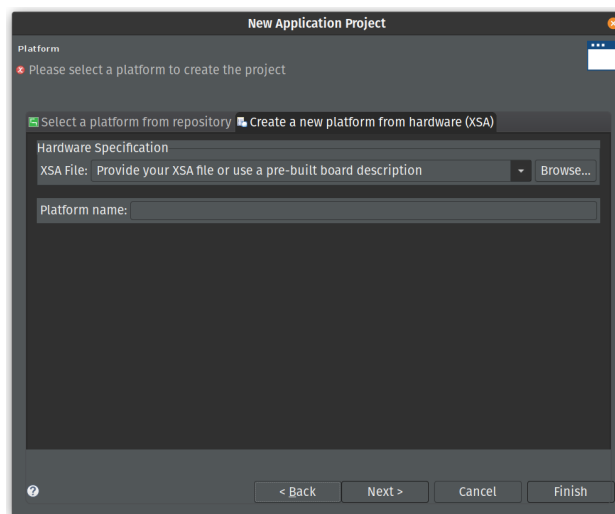


Figure 3: For the platform, we need to get our build Kria configuration. In the "Create a new platform" tab, click the "Browse..." button.

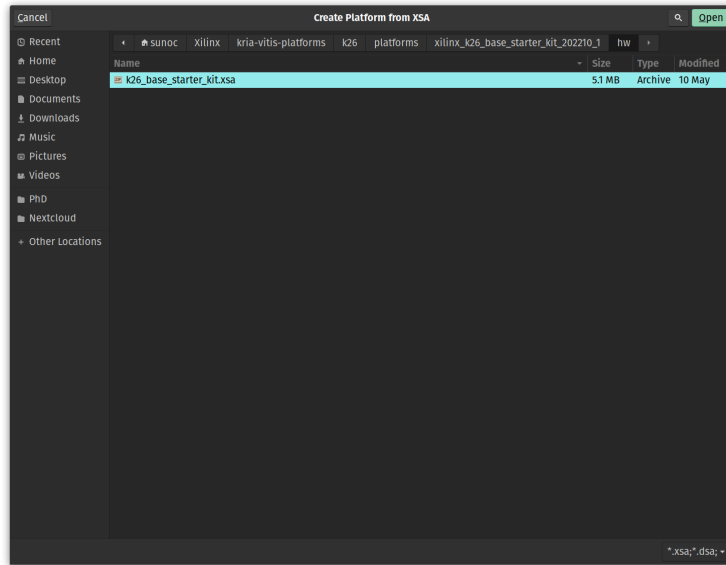


Figure 4: In the file explorer, we should navigate in the "k26" directory, where the configuration file was build. From here we are looking for a ".xsa" file, located in a "hw" directory, as visible.

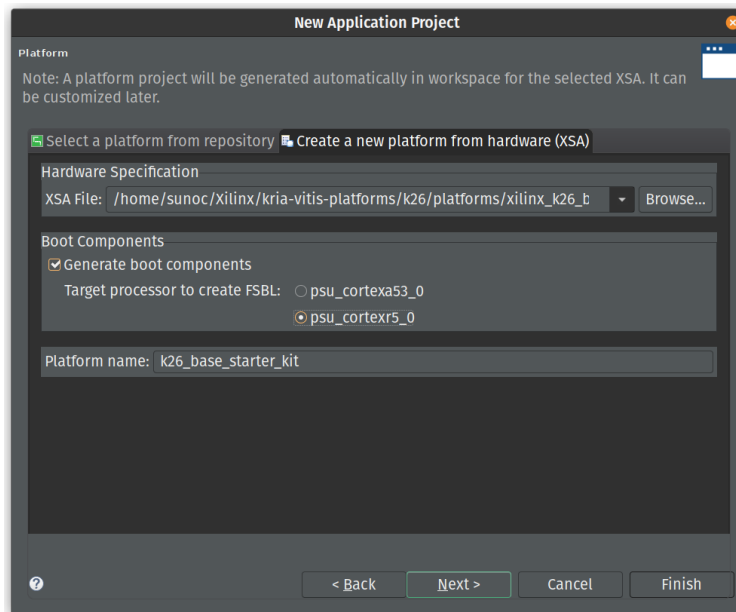


Figure 5: With the configuration file loaded, we can now select a name for our platform, but most importantly, we have to select the "psu Cortex5 0" core as a target. The other, Cortex 53 is the APU running Linux.

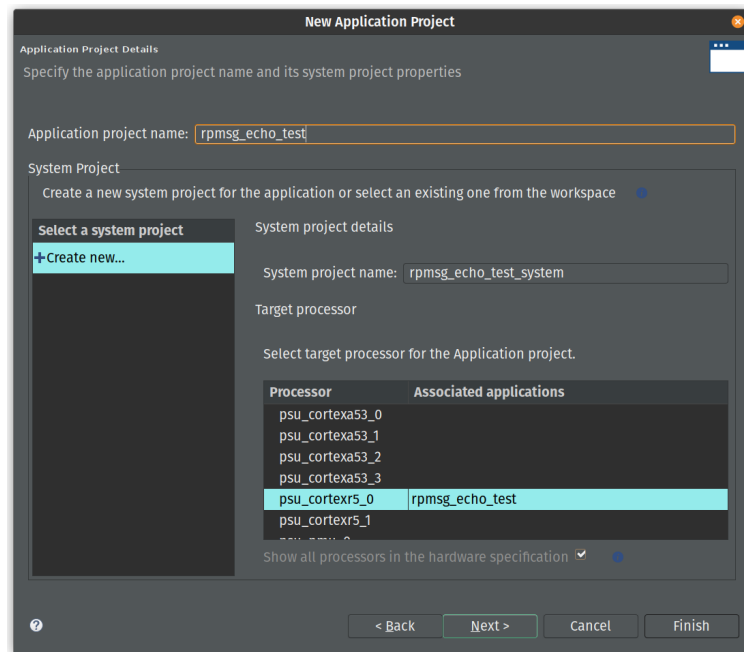


Figure 6: In this next window, we can give a name to our firmware project. It is also critical here to select the core we want to build for. Once again, we want to use the "psu cortex5 0".

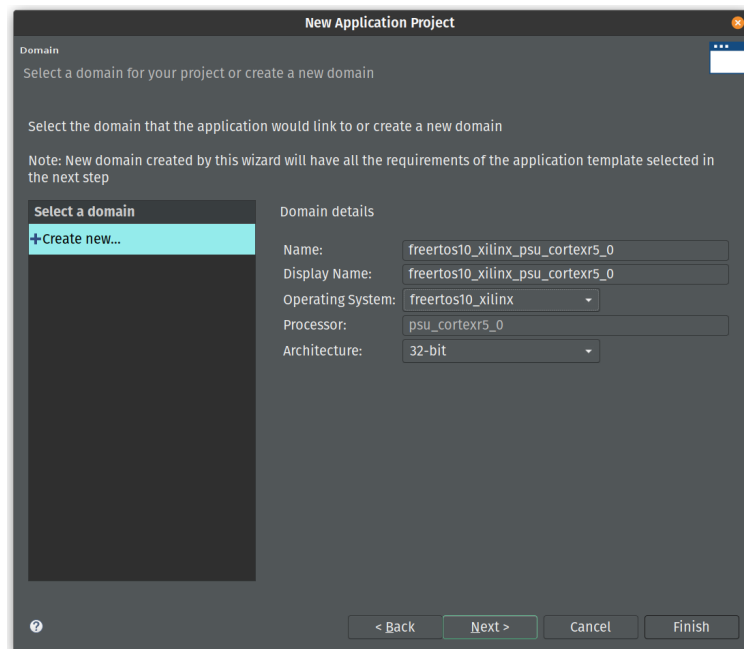


Figure 7: Here, we want to select "freertos10 xilinx" as our Operating System. The rest can remain unchanged.

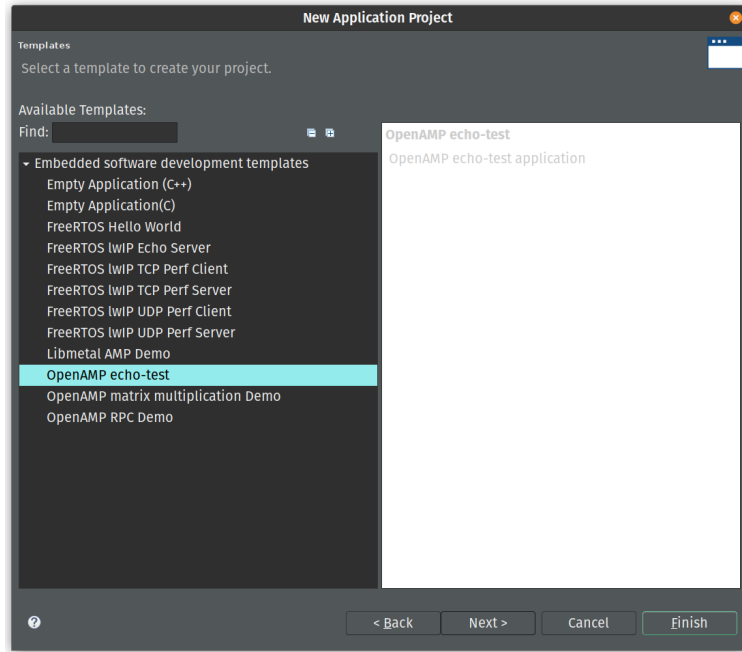


Figure 8: Finally, we can select the demonstration template we are going to use; here we go with "OpenAMP echo-test" since we want to have some simple try of the RPMsg system. Finish.

In the Xilinx documentation, it is made mention of the addresses setting that should be checked in the `script.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:

1	<code>psu_ddr_S_AXI_BASEADDR</code>	<code>0x3ed00000</code>
2	<code>psu_ocm_ram_1_S_AXI_BASEADDR</code>	<code>0xfffc0000</code>
3	<code>psu_r5_tcm_ram_0_S_AXI_BASEADDR</code>	<code>0x00000000</code>
4	<code>psu_r5_tcm_ram_1_S_AXI_BASEADDR</code>	<code>0x00020000</code>

6 Building a real-time firmware

6.1 Setting up Vitis IDE

7 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 !

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

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

8 Building micro-ROS as a static library

```
1 pushd /home/$USER/Downloads
2 wget https://developer.arm.com/-/media/Files/downloads/\
3 gnu/12.2.mpacbti-rel1/binrel/arm-gnu-toolchain-12.2\
4 .mpacbti-rel1-x86_64-arm-none-eabi.tar.xz
5 tar -xvf arm-gnu-toolchain-12.2.mpacbti-rel1-x86_64-\
6 arm-none-eabi.tar.xz
7 popd
8
9 toolchain="/home/$USER/Downloads/arm-gnu-toolchain-\
10 12.2.mpacbti-rel1-x86_64-arm-none-eabi/"
11
12
13 docker run -d --name ros_build -it --net=host \
14 --hostname ros_build \
15 -v /dev:/dev \
16 -v $toolchain:/armr5-toolchain \
17 --privileged ros:iron
18
19 docker exec -it ros_build bash
```

- 9 Adding micro-ROS to a firmware project
- 10 Loading a real-time firmware
- 11 Running a ROS2 node
 - 11.1 On the host Linux
 - 11.2 In a container
- 12 micro-ROS agent

A DTO patch

This file is available in this repository: system.patch

```
1 diff -u --label /ssh\:kria\:/home/ubuntu/system_original.dts --label
2 ↪ /ssh\:kria\:/home/ubuntu/system.dts /tmp/tramp.KJbEbZ.dts /tmp/tramp.zHBG7v.dts
3 --- /ssh\:kria\:/home/ubuntu/system_original.dts
4 +++ /ssh\:kria\:/home/ubuntu/system.dts
5 @@ -138,7 +138,7 @@
6
7     firmware {
8
9         -            zynqmp-firmware {
10        +            zynqmp_firmware: zynqmp-firmware {
11                compatible = "xlnx,zynqmp-firmware";
12                #power-domain-cells = <0x01>;
13                method = "smc";
14        @@ -719,7 +719,7 @@
15                phandle = <0x44>;
16            };
17
18        -            interrupt-controller@f9010000 {
19        +            gic: interrupt-controller@f9010000 {
20                compatible = "arm,gic-400";
21                #interrupt-cells = <0x03>;
22                reg = <0x00 0xf9010000 0x00 0x10000 0x00 0xf9020000 0x00
23        ↪ 0x20000 0x00 0xf9040000 0x00 0x20000 0x00 0xf9060000 0x00 0x20000>;
24        @@ -1536,7 +1536,7 @@
25                pinctrl-names = "default";
26                u-boot,dm-pre-reloc;
27                compatible = "xlnx,zynqmp-uart\0cdns,uart-rip12";
28        -            status = "okay";
29        +            status = "disabled";
30                interrupt-parent = <0x04>;
31                interrupts = <0x00 0x16 0x04>;
32                reg = <0x00 0xff010000 0x00 0x1000>;
33        @@ -1909,6 +1909,84 @@
34                pwms = <0x1b 0x02 0x9c40 0x00>;
35            };
36
37        +            reserved-memory {
38        +                #address-cells = <2>;
39        +                #size-cells = <2>;
40        +                ranges;
41        +                rpu0vdev0vring0: rpu0vdev0vring0@3ed40000 {
42        +                    no-map;
43        +                    reg = <0x0 0x3ed40000 0x0 0x4000>;
44        +                };
45        +                rpu0vdev0vring1: rpu0vdev0vring1@3ed44000 {
46        +                    no-map;
47        +                    reg = <0x0 0x3ed44000 0x0 0x4000>;
48        +                };
49        +                rpu0vdev0buffer: rpu0vdev0buffer@3ed48000 {
50        +                    no-map;
51        +                    reg = <0x0 0x3ed48000 0x0 0x100000>;
```



```

50 +         };
51 +         rproc_0_reserved: rproc_0_reserved@3ed00000 {
52 +             no-map;
53 +             reg = <0x0 0x3ed00000 0x0 0x40000>;
54 +         };
55 +     };
56 +     tcm_0a: tcm_0a@ffe00000 {
57 +         no-map;
58 +         reg = <0x0 0xffe00000 0x0 0x10000>;
59 +         status = "okay";
60 +         compatible = "mmio-sram";
61 +         power-domain = <&zynqmp_firmware 15>;
62 +     };
63 +     tcm_0b: tcm_0b@ffe20000 {
64 +         no-map;
65 +         reg = <0x0 0xffe20000 0x0 0x10000>;
66 +         status = "okay";
67 +         compatible = "mmio-sram";
68 +         power-domain = <&zynqmp_firmware 16>;
69 +     };
70 +     rf5ss@ff9a0000 {
71 +         compatible = "xlnx,zynqmp-r5-remoteproc";
72 +         xlnx,cluster-mode = <1>;
73 +         ranges;
74 +         reg = <0x0 0xFF9A0000 0x0 0x10000>;
75 +         #address-cells = <0x2>;
76 +         #size-cells = <0x2>;
77 +         r5f_0 {
78 +             compatible = "xilinx,r5f";
79 +             #address-cells = <2>;
80 +             #size-cells = <2>;
81 +             ranges;
82 +             sram = <&tcm_0a &tcm_0b>;
83 +             memory-region = <&rproc_0_reserved>, <&rpu0vdev0buffer>,
↵ <&rpu0vdev0vring0>, <&rpu0vdev0vring1>;
84 +             power-domain = <&zynqmp_firmware 7>;
85 +             mboxs = <&ipi_mailbox_rpu0 0>, <&ipi_mailbox_rpu0 1>;
86 +             mbox-names = "tx", "rx";
87 +         };
88 +     };
89 +     zynqmp_ipi1 {
90 +         compatible = "xlnx,zynqmp-ipi-mailbox";
91 +         interrupt-parent = <&gic>;
92 +         interrupts = <0 29 4>;
93 +         xlnx,ipi-id = <7>;
94 +         #address-cells = <1>;
95 +         #size-cells = <1>;
96 +         ranges;
97 +         /* APU<->RPU0 IPI mailbox controller */
98 +         ipi_mailbox_rpu0: mailbox@ff990600 {
99 +             reg = <0xff990600 0x20>,
100 +                 <0xff990620 0x20>,
101 +                 <0xff9900c0 0x20>,
102 +                 <0xff9900e0 0x20>;

```

```
103 +         reg-names = "local_request_region",
104 +         "local_response_region",
105 +         "remote_request_region",
106 +         "remote_response_region";
107 +     #mbox-cells = <1>;
108 +     xlnx,ipi-id = <1>;
109 +     };
110 + };
111 +
112 +
113 +     __symbols__ {
114 +         cpu0 = "/cpus/cpu@0";
115 +         cpu1 = "/cpus/cpu@1";
116 +
117 + Diff finished.  Wed May 24 10:15:49 2023
```

B Custom toolchain CMake settings

This file is available in this repository: custom_r5f_toolchain.cmake

```
1 set(CMAKE_SYSTEM_NAME Generic)
2 set(CMAKE_CROSSCOMPILING 1)
3 set(CMAKE_TRY_COMPILE_TARGET_TYPE STATIC_LIBRARY)
4 set(CMAKE_INSTALL_LIBDIR /usr/)
5 set(PLATFORM_NAME "LwIP")
6
7
8 set(ARCH_CPU_FLAGS "-mcpu=cortex-r5 -mthumb -mfpv3-d16 -mfloat-abi=hard -DARMv5 -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)
9 set(ARCH_OPT_FLAGS "")
10
11 set(CMAKE_C_COMPILER arm-none-eabi-gcc)
12 set(CMAKE_CXX_COMPILER arm-none-eabi-g++)
13
14 set(CMAKE_C_FLAGS_INIT "-std=c11 ${ARCH_CPU_FLAGS} ${ARCH_OPT_FLAGS}
  ↳ -DCLOCK_MONOTONIC=0" CACHE STRING "" FORCE)
15 set(CMAKE_CXX_FLAGS_INIT "-std=c++14 ${ARCH_CPU_FLAGS} ${ARCH_OPT_FLAGS}
  ↳ -DCLOCK_MONOTONIC=0" CACHE STRING "" FORCE)
16
17
18
19 set(__BIG_ENDIAN__ 0)
```

C Custom Colcon meta settings

This file is available in this repository: `custom r5f colcon.meta`

```
1 {
2   "names": {
3     "tracetools": {
4       "cmake-args": [
5         "-DTRACETOOLS_DISABLED=ON",
6         "-DTRACETOOLS_STATUS_CHECKING_TOOL=OFF"
7       ]
8     },
9     "rosidl_typesupport": {
10      "cmake-args": [
11        "-DROSIDL_TYPESUPPORT_SINGLE_TYPESUPPORT=ON"
12      ]
13    },
14    "rcl": {
15      "cmake-args": [
16        "-DBUILD_TESTING=OFF",
17        "-DRCL_COMMAND_LINE_ENABLED=OFF",
18        "-DRCL_LOGGING_ENABLED=OFF"
19      ]
20    },
21    "rcutils": {
22      "cmake-args": [
23        "-DENABLE_TESTING=OFF",
24        "-DRCUTILS_NO_FILESYSTEM=ON",
25        "-DRCUTILS_NO_THREAD_SUPPORT=ON",
26        "-DRCUTILS_NO_64_ATOMIC=ON",
27        "-DRCUTILS_AVOID_DYNAMIC_ALLOCATION=ON"
28      ]
29    },
30    "microxrcedds_client": {
31      "cmake-args": [
32        "-DUCCLIENT_PIC=OFF",
33        "-DUCCLIENT_PROFILE_UDP=OFF",
34        "-DUCCLIENT_PROFILE_TCP=OFF",
35        "-DUCCLIENT_PROFILE_DISCOVERY=OFF",
36        "-DUCCLIENT_PROFILE_SERIAL=OFF",
37        "-DUCCLIENT_PROFILE_STREAM_FRAMING=ON",
38        "-DUCCLIENT_PROFILE_CUSTOM_TRANSPORT=ON"
39      ]
40    },
41    "rmw_microxrcedds": {
42      "cmake-args": [
43        "-DRMW_UXRCE_MAX_NODES=1",
44        "-DRMW_UXRCE_MAX_PUBLISHERS=5",
45        "-DRMW_UXRCE_MAX_SUBSCRIPTIONS=5",
46        "-DRMW_UXRCE_MAX_SERVICES=1",
47        "-DRMW_UXRCE_MAX_CLIENTS=1",
48        "-DRMW_UXRCE_MAX_HISTORY=4",
49        "-DRMW_UXRCE_TRANSPORT=custom"
50      ]
51    }
52  }
```

52

}

53

}

D Firmware time functions

D.1 main

This file is available in this repository: clock.c

```
1  #include "microros.h"
2
3
4  int _gettimeofday( struct timeval *tv, void *tzvp )
5  {
6      XTime t = 0;
7      XTime_GetTime(&t); //get uptime in nanoseconds
8      tv->tv_sec = t / 1000000000; // convert to seconds
9      tv->tv_usec = ( t % 1000000000 ) / 1000; // get remaining microseconds
10     return 0; // return non-zero for error
11 } // end _gettimeofday()
12
13
14 void UTILS_NanosecondsToTimespec( int64_t llSource,
15                                   struct timespec * const pxDestination )
16 {
17     long lCarrySec = 0;
18
19     /* Convert to timespec. */
20     pxDestination->tv_sec = ( time_t ) ( llSource / NANoseconds_PER_SECOND );
21     pxDestination->tv_nsec = ( long ) ( llSource % NANoseconds_PER_SECOND );
22
23     /* Subtract from tv_sec if tv_nsec < 0. */
24     if( pxDestination->tv_nsec < 0L )
25     {
26         /* Compute the number of seconds to carry. */
27         lCarrySec = ( pxDestination->tv_nsec / ( long ) NANoseconds_PER_SECOND ) + 1L;
28
29         pxDestination->tv_sec -= ( time_t ) ( lCarrySec );
30         pxDestination->tv_nsec += lCarrySec * ( long ) NANoseconds_PER_SECOND;
31     }
32 }
33
34 int clock_gettime( clockid_t clock_id,
35                   struct timespec * tp )
36 {
37     TimeOut_t xCurrentTime = { 0 };
38
39     /* Intermediate variable used to convert TimeOut_t to struct timespec.
40      * Also used to detect overflow issues. It must be unsigned because the
41      * behavior of signed integer overflow is undefined. */
42     uint64_t ullTickCount = 0ULL;
43
44     /* Silence warnings about unused parameters. */
45     ( void ) clock_id;
46
47     /* Get the current tick count and overflow count. vTaskSetTimeOutState()
48      * is used to get these values because they are both static in tasks.c. */
49     vTaskSetTimeOutState( &xCurrentTime );
```

```

50
51     /* Adjust the tick count for the number of times a TickType_t has overflowed.
52      * portMAX_DELAY should be the maximum value of a TickType_t. */
53     ullTickCount = ( uint64_t ) ( xCurrentTime.xOverflowCount ) << ( sizeof( TickType_t
↪ ) * 8 );
54
55     /* Add the current tick count. */
56     ullTickCount += xCurrentTime.xTimeOnEntering;
57
58     /* Convert ullTickCount to timespec. */
59     UTILS_NanosecondsToTimespec( ( int64_t ) ullTickCount * NANoseconds_PER_TICK, tp );
60
61     return 0;
62 }

```

D.2 header file

```

1  /**< Microseconds per second. */
2  #define MICROSECONDS_PER_SECOND    ( 1000000LL )
3  /**< Nanoseconds per second. */
4  #define NANoseconds_PER_SECOND     ( 1000000000LL )
5  /**< Nanoseconds per FreeRTOS tick. */
6  #define NANoseconds_PER_TICK       ( NANoseconds_PER_SECOND / configTICK_RATE_HZ )

```

E Firmware memory allocation functions

E.1 main

This file is available in this repository: allocators.c

```
1  #include "allocators.h"
2
3  //int absoluteUsedMemory = 0;
4  //int usedMemory = 0;
5
6  void * __freertos_allocate(size_t size, void * state){
7      (void) state;
8      LPRINTF("-- Alloc %d (prev: %d B)\r\n",size, xPortGetFreeHeapSize());
9      // absoluteUsedMemory += size;
10     // usedMemory += size;
11
12     LPRINTF("Return for the allocate function w parameter size = %d\r\n", size);
13
14     return pvPortMalloc(size);
15 }
16
17 void __freertos_deallocate(void * pointer, void * state){
18     (void) state;
19     LPRINTF("-- Free 0x%x (prev: %d B)\r\n", pointer, xPortGetFreeHeapSize());
20     if (NULL != pointer)
21     {
22         // LPRINTF("Pointer is not null.\r\n");
23         // usedMemory -= getBlockSize(pointer);
24         // LPRINTF("usedMemory var updated: %d\r\n", usedMemory);
25         vPortFree(pointer);
26     }
27     else
28     {
29         LPERROR("Trying to deallocate a null pointed. Doing nothing.\r\n");
30     }
31 }
32
33 void * __freertos_reallocate(void * pointer, size_t size, void * state){
34     (void) state;
35     LPRINTF("-- Realloc 0x%x -> %d (prev: %d B)\r\n", pointer, size,
36     ↪ xPortGetFreeHeapSize());
37     // absoluteUsedMemory += size;
38     // usedMemory += size;
39     if (NULL == pointer)
40     {
41         return __freertos_allocate(size, state);
42     }
43     else
44     {
45         // usedMemory -= getBlockSize(pointer);
46         //
47         // return pvPortRealloc(pointer,size);
48
49         __freertos_deallocate(pointer, state);
```



```

49         return __freertos_allocate(size, state);
50
51     }
52 }
53
54 void * __freertos_zero_allocate(size_t number_of_elements, size_t size_of_element, void
↳ * state){
55     (void) state;
56     LPRINTF("-- Calloc %d x %d = %d -> (prev: %d
↳ B)\r\n", number_of_elements, size_of_element, number_of_elements*size_of_element,
↳ xPortGetFreeHeapSize());
57     // absoluteUsedMemory += number_of_elements*size_of_element;
58     // usedMemory += number_of_elements*size_of_element;
59
60     return pvPortCalloc(number_of_elements, size_of_element);
61 }

```

E.2 header file

```

1  #ifndef _ALLOCATORS_H_
2  #define _ALLOCATORS_H_
3
4  #include "microros.h"
5
6  extern int absoluteUsedMemory;
7  extern int usedMemory;
8
9
10 void * __freertos_allocate(size_t size, void * state);
11 void __freertos_deallocate(void * pointer, void * state);
12 void * __freertos_reallocate(void * pointer, size_t size, void * state);
13 void * __freertos_zero_allocate(size_t number_of_elements,
14 size_t size_of_element, void * state);
15
16 #endif // _ALLOCATORS_H_

```