KRIA board Vitis IDE setup

Vincent Conus[*]

2024-01-31

Contents

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[1]. You will need to get a file named something like `Xilinx_Unified_2022.2_1014_8888_Lin64.bin`[2].

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.

1.1 Dependencies & installation

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

```
sudo apt-get -y update

sudo apt-get -y install libncurses-dev \
    ncurses-term \
    ncurses-base \
```

[*]vincent.conus@protonmail.com

[1]https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vitis.html

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

```
    ncurses-bin \
    libncurses5 \
    libtinfo5 \
    libncurses5-dev \
    libncursesw5-dev
```

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

```
./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:

```
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.

1.2 Platform configuration file

With a Xilinx account, a `.bsp` archive can be downloaded for the taget platform[3].

Once the file is downloaded, the following commands allows to "un-tar" it, making th needed `.xsa` file accessible via a file explorer.

```
tar xvfz xilinx-kr260-starterkit-v2022.2-10141622.bsp
ls xilinx-kr260-starterkit-2022.2/hardware/xilinx-kr260-starterkit-2022.2/
```

1.3 [DEPRECATED] 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[4].

A Xilinx dedicated repository[5] 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[6], 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:

```
sudo apt-get update
sudo apt-get install cmake tcl libidn11-dev \
    libidn-dev libidn12 idn
```

[3]https://www.xilinx.com/member/forms/download/xef.html?filename=xilinx-kr260-starterkit-v2022.2-10141622.bsp

[4]https://xilinx.github.io/kria-apps-docs/kv260/2022.1/build/html/docs/build_vitis_platform.html?highlight=xsa

[5]https://github.com/Xilinx/kria-vitis-platforms

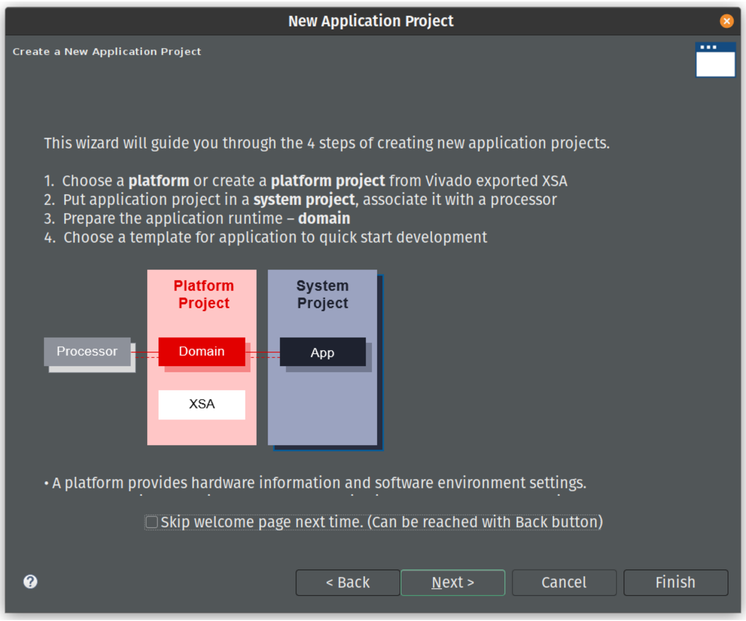
[6]https://support.xilinx.com/s/question/0D52E00006jrzsYSAQ/platform-project-cannot-be-created-on-vitis?language=en_US

```
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
```

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.



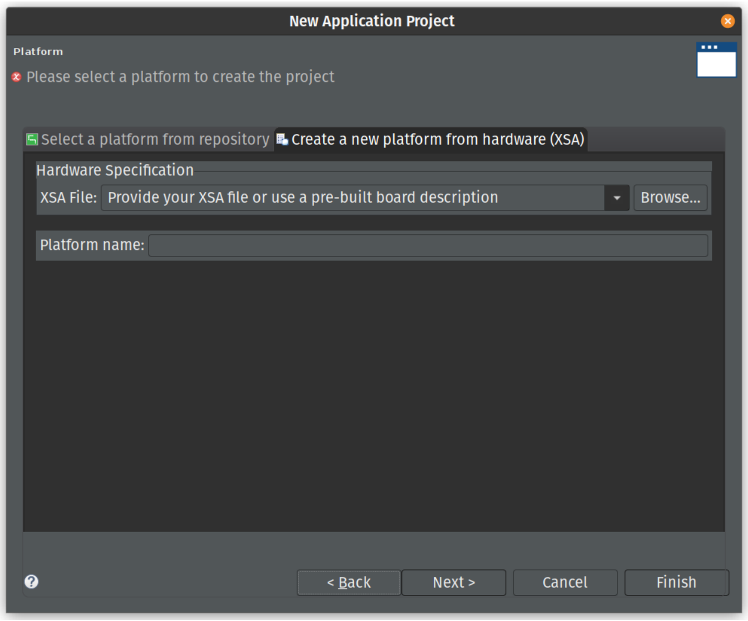
This wizard will guide you through the 4 steps of creating new application projects.

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


New Application Project

Platform

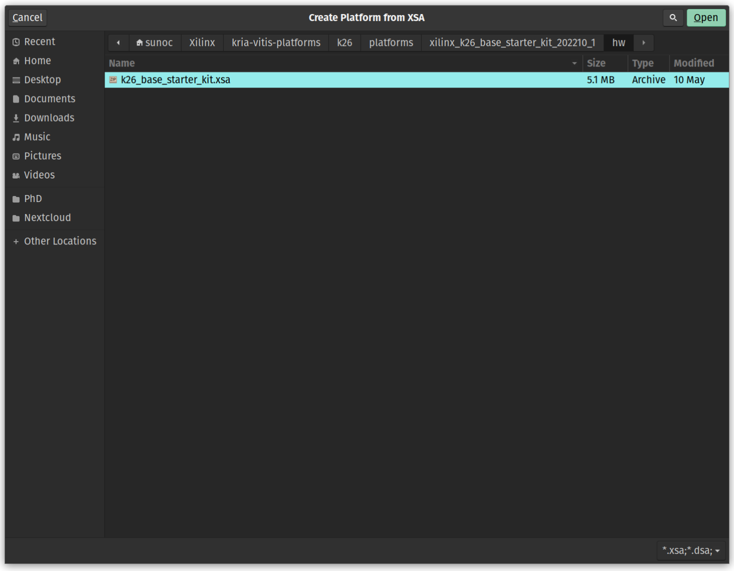
Please select a platform to create the project

Select a platform from repository
Create a new platform from hardware (XSA)
Hardware Specification
Browse...
< Back
Next >
Cancel
Finish

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

Platform name:

Figure 2: For the platform, we need to get our build Kria configuration. In the "Create a new platform" tab, click the "Browse..." button.

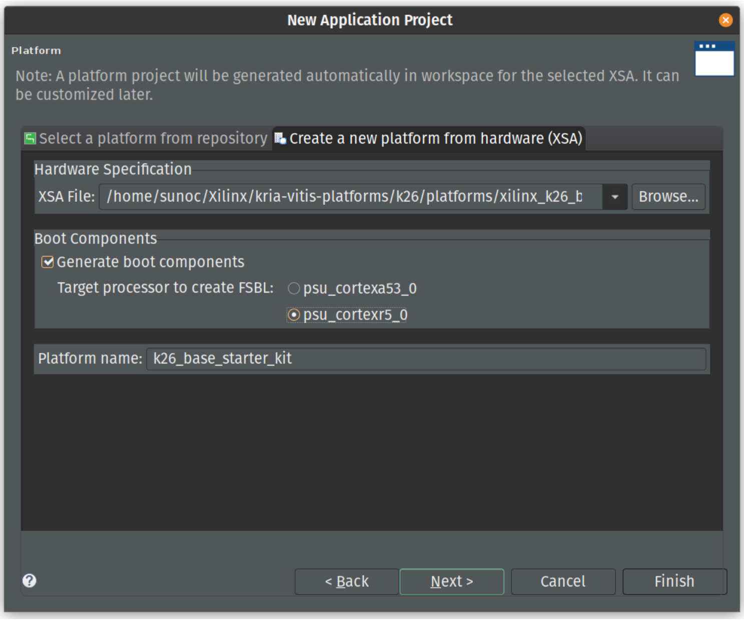

Create Platform from XSA

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 3: 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

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

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

Hardware Specification

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

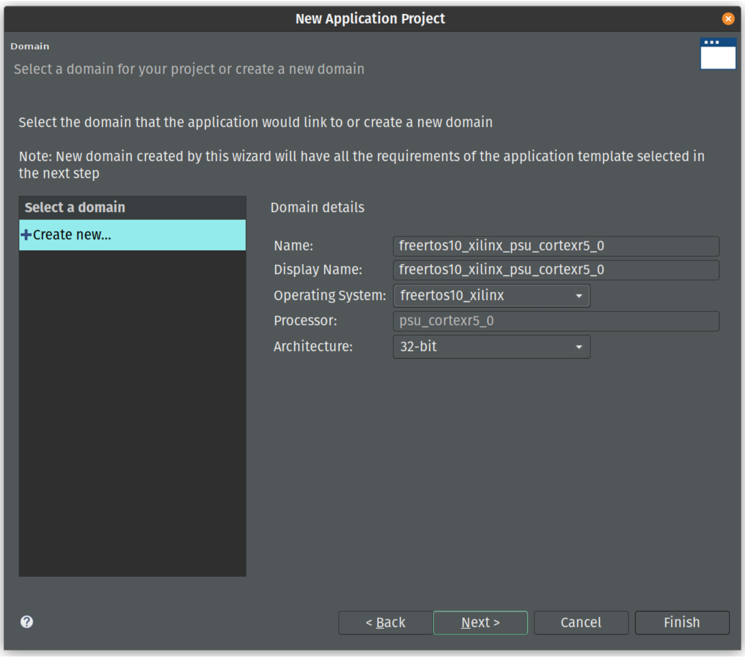
Boot Components

☑ Generate boot components

Target processor to create FSBL: ○ psu_cortexa53_0 ● psu_cortexr5_0

Platform name: k26_base_starter_kit

Figure 4: 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

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

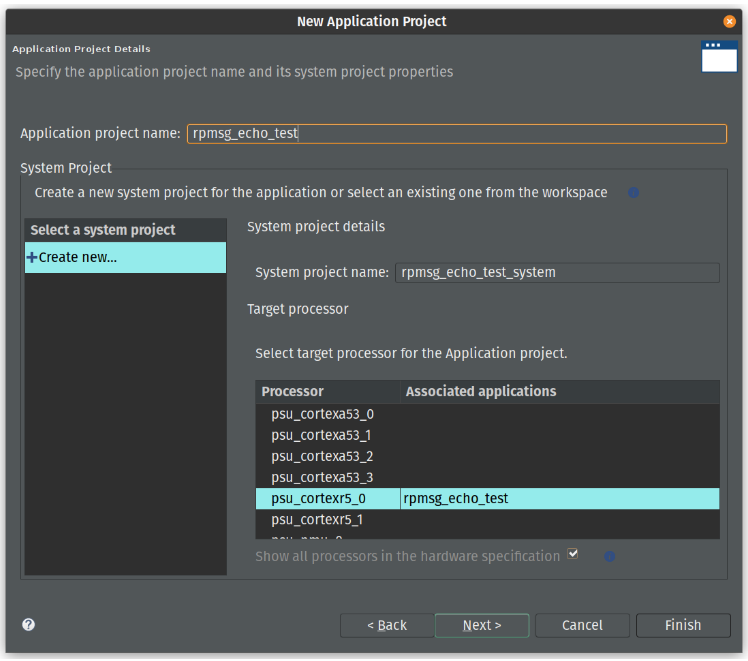
Note: New domain created by this wizard will have all the requirements of the application template selected in the next step

Select a domain

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


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

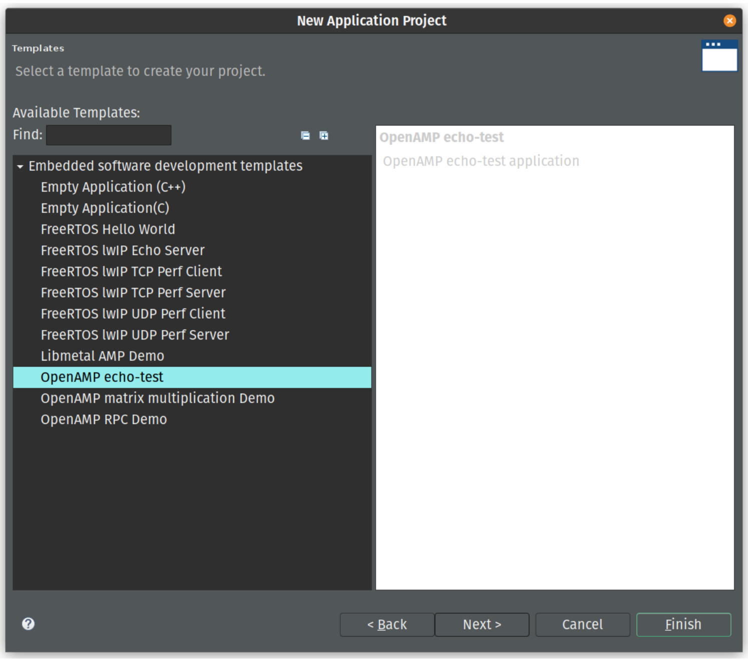
Target processor

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

Templates

Select a template to create your project.

Available Templates:

Find:

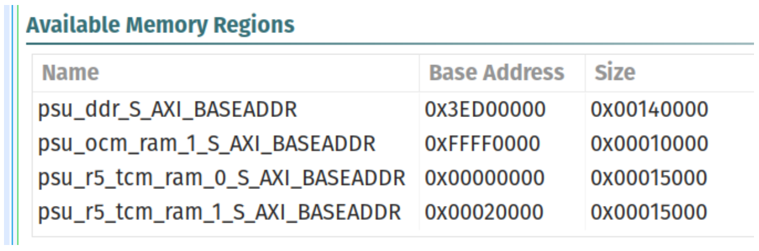
- Embedded software development templates
  - Empty Application (C++)
  - Empty Application(C)
  - FreeRTOS Hello World
  - FreeRTOS lwIP Echo Server
  - FreeRTOS lwIP TCP Perf Client
  - FreeRTOS lwIP TCP Perf Server
  - FreeRTOS lwIP UDP Perf Client
  - FreeRTOS lwIP UDP Perf Server
  - Libmetal AMP Demo
  - OpenAMP echo-test
  - OpenAMP matrix multiplication Demo
  - OpenAMP RPC Demo

OpenAMP echo-test

OpenAMP echo-test application

Figure 7: 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. The values in the figure 8 below 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, including the new DTO patch without overlapping memory:



Available Memory Regions

Name	Base Address	Size
psu_ddr_S_AXI_BASEADDR	0X3ED00000	0X00140000
psu_ocm_ram_1_S_AXI_BASEADDR	0xFFFF0000	0X00010000
psu_r5_tcm_ram_0_S_AXI_BASEADDR	0x00000000	0X00015000
psu_r5_tcm_ram_1_S_AXI_BASEADDR	0x00020000	0X00015000

Figure 8: lscript.ld memory configuration for the firmware memory setup. The same file is available as a whole in this repository's src directory.

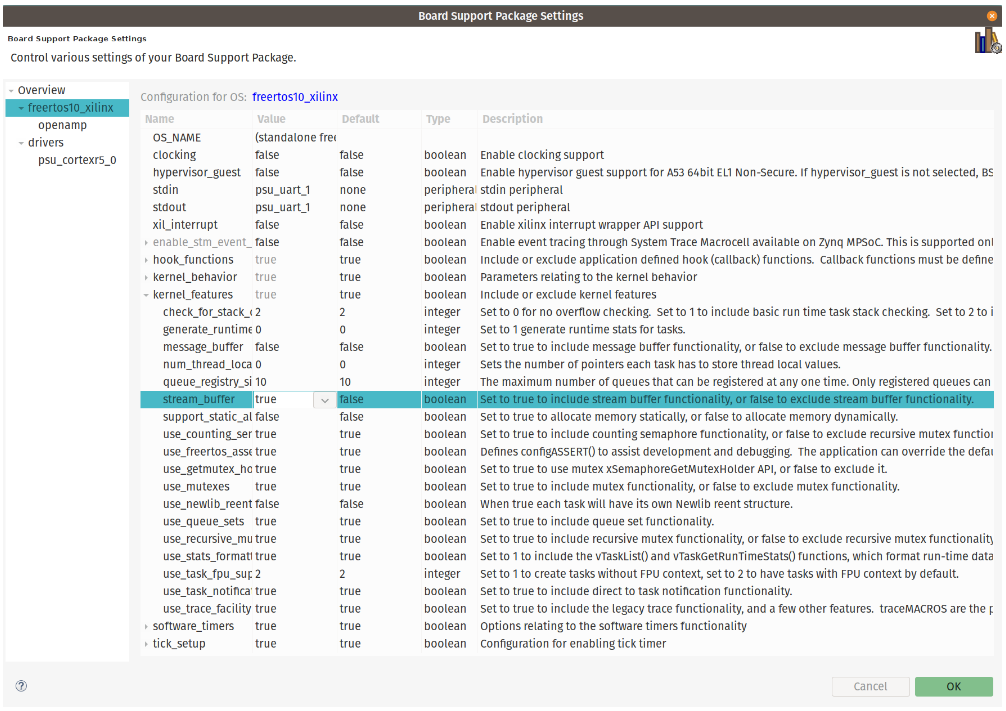
Once your example project is built and you have a `.elf` file available, you can jump directly in further sections to see how to deploy and use your firmware.

The section in between will present setup specifically needed for micro-ROS.

3 Enabling the Stream Buffer system

This is a subpart in the general configuration in the project related to some specific functions for FreeRTOS threads messaging system, however, this point in particular created so much pain I needed to include in early in this guide for not to forget about it and keeping a clear track on how to enable this setting.

Indeed, two settings need to be enabled in order to be able to call functions such as `xMessageBufferCreate`, useful when working with tasks in FreeRTOS, as visible in the figure 9 below:



Board Support Package Settings

Board Support Package Settings

Control various settings of your Board Support Package.

Configuration for OS: freertos10_xilinx

Name	Value	Default	Type	Description
OS_NAME	(standalone fre			
clocking	false	false	boolean	Enable clocking support
hypervisor_guest	false	false	boolean	Enable hypervisor guest support for A53 64bit EL1 Non-Secure. If hypervisor_guest is not selected, BS
stdin	psu_uart_1	none	peripheral	stdin peripheral
stdout	psu_uart_1	none	peripheral	stdout peripheral
xil_interrupt	false	false	boolean	Enable xilinx interrupt wrapper API support
enable_stm_event_	false	false	boolean	Enable event tracing through System Trace Macrocell available on Zynq MPSoC. This is supported onl
hook_functions	true	true	boolean	Include or exclude application defined hook (callback) functions. Callback functions must be define
kernel_behavior	true	true	boolean	Parameters relating to the kernel behavior
kernel_features	true	true	boolean	Include or exclude kernel features
check_for_stack_	2	2	integer	Set to 0 for no overflow checking. Set to 1 to include basic run time task stack checking. Set to 2 to i
generate_runtime	0	0	integer	Set to 1 generate runtime stats for tasks.
message_buffer	false	false	boolean	Set to true to include message buffer functionality, or false to exclude message buffer functionality.
num_thread_loca	0	0	integer	Sets the number of pointers each task has to store thread local values.
queue_registry_si	10	10	integer	The maximum number of queues that can be registered at any one time. Only registered queues can
stream_buffer	true	false	boolean	Set to true to include stream buffer functionality, or false to exclude stream buffer functionality.
support_static_al	false	false	boolean	Set to true to allocate memory statically, or false to allocate memory dynamically.
use_counting_ser	true	true	boolean	Set to true to include counting semaphore functionality, or false to exclude recursive mutex functio
use_freertos_asse	true	true	boolean	Defines configASSERT() to assist development and debugging. The application can override the defa
use_getmutex_ho	true	true	boolean	Set to true to use mutex xSemaphoreGetMutexHolder API, or false to exclude it.
use_mutexes	true	true	boolean	Set to true to include mutex functionality, or false to exclude mutex functionality.
use_newlib_reent	false	false	boolean	When true each task will have its own Newlib reent structure.
use_queue_sets	true	true	boolean	Set to true to include queue set functionality.
use_recursive_mu	true	true	boolean	Set to true to include recursive mutex functionality, or false to exclude recursive mutex functionalit
use_stats_format	true	true	boolean	Set to 1 to include the vTaskList() and vTaskGetRunTimeStats() functions, which format run-time data
use_task_fpu_sup	2	2	integer	Set to 1 to create tasks without FPU context, set to 2 to have tasks with FPU context by default.
use_task_notifica	true	true	boolean	Set to true to include direct to task notification functionality.
use_trace_facility	true	true	boolean	Set to true to include the legacy trace functionality, and a few other features. traceMACROS are the
software_timers	true	true	boolean	Options relating to the software timers functionality
tick_setup	true	true	boolean	Configuration for enabling tick timer

Figure 9: Enabling Stream Buffer in the Vitis IDE setting: this is a setting that can be found in the "platform.spr" element of your project (the platform, not the firmware project itself). From that file, you can access the settings with the button "Modify BSP Settings", and then as visible, in the tab `freertos10_xilinx`, it is needed to toggle here the `stream_buffer` setting in the `kernel_features`, from the default "false" to "true".

The second setting is useful in the case when a buffer callback function is used, such as `xMessageBufferCreateWithCallback`. In that case, you must include `#define configUSE_SB_COMPLETED_CALLBACK 1` on the top of you header file (in our project, this will happen in the `microros.h` header file), before the `#include "FreeRTOS.h"` in order to override the setting from this include.

4 Including micro-ROS to the real-time firmware

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.

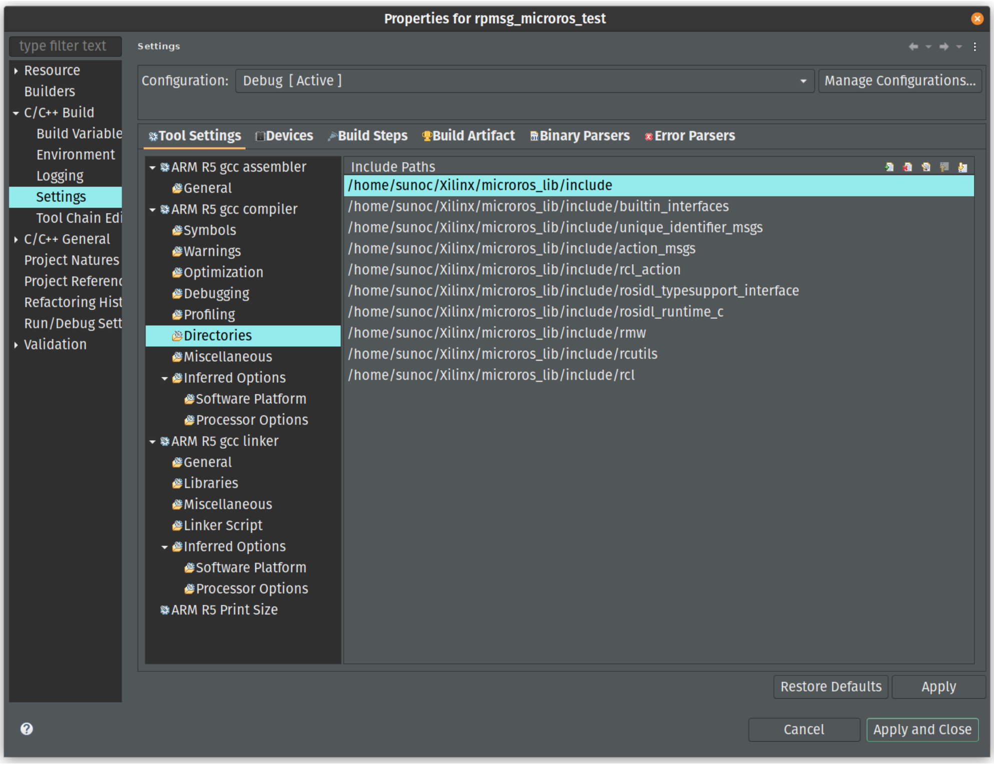
On the host machine running the IDE, we can download the static library and the include files from the Docker builder. Here, we assume your Vitis IDE workspace sits in you home directory, at `~/workspace`, and that the Docker container is named `ros_build`:

```
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 tool-chain, but the figures 10 and 11 below will show you a setup that worked to have the IDE to recognize the include files and to be able to use them for compiling the firmware.

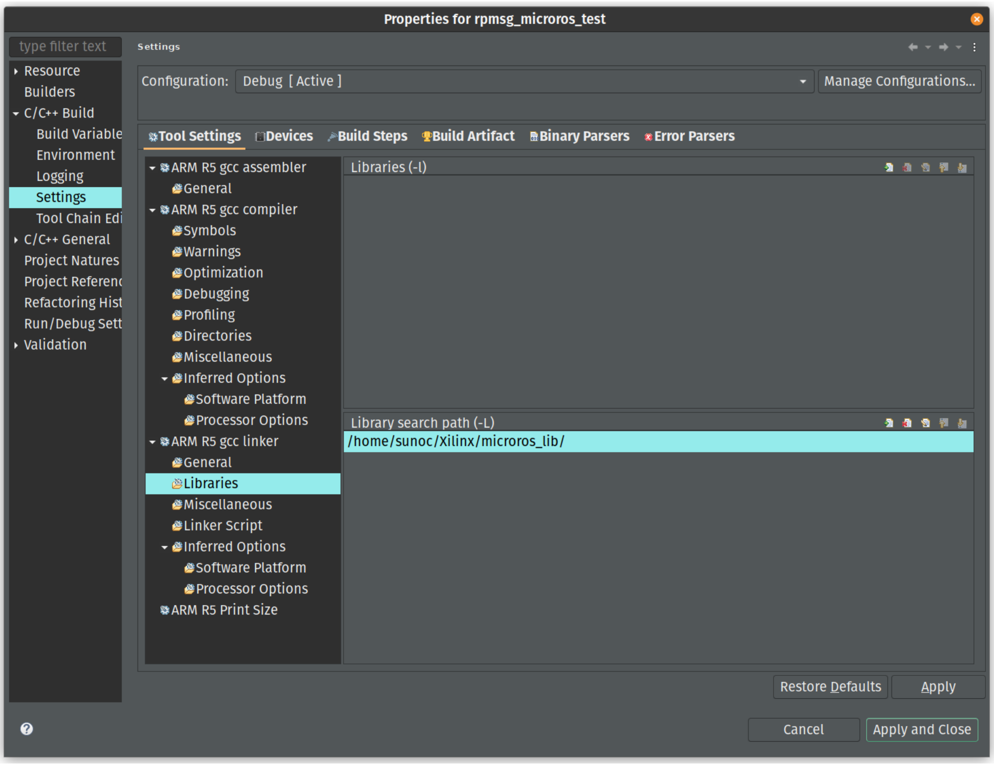


Configuration: Debug [Active]

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 10: 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, as visible in this figure.



Configuration: Debug [Active]

Libraries (-l)

Library search path (-L)

/home/sunoc/Xilinx/microros_lib/

Figure 11: Secondly, in the gcc linker "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 and as a minimal test to see if the setup was made correctly, 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.