# **QT 5.15.2 CROSS COMPILATION FOR RASPBERRY**

1. **Raspberry Pi4 OS and VNC server setup:**

Install Raspbian OS on a Raspberry Pi 4 and install the VNC server used to share graphical desktop system. follow below link to setup Raspbian os and vnc server.

<https://www.instructables.com/How-to-Setup-Raspberry-Pi-Without-Monitor-and-Keyb/>

1. **QT 5.15.2 ON UBUNTU 20 LTS or 18 LTS:**

Install ubuntu 18 or 20 LTS on oracle VM VirtualBox and install Qt 5.15.2 and follow below steps for cross compilation.

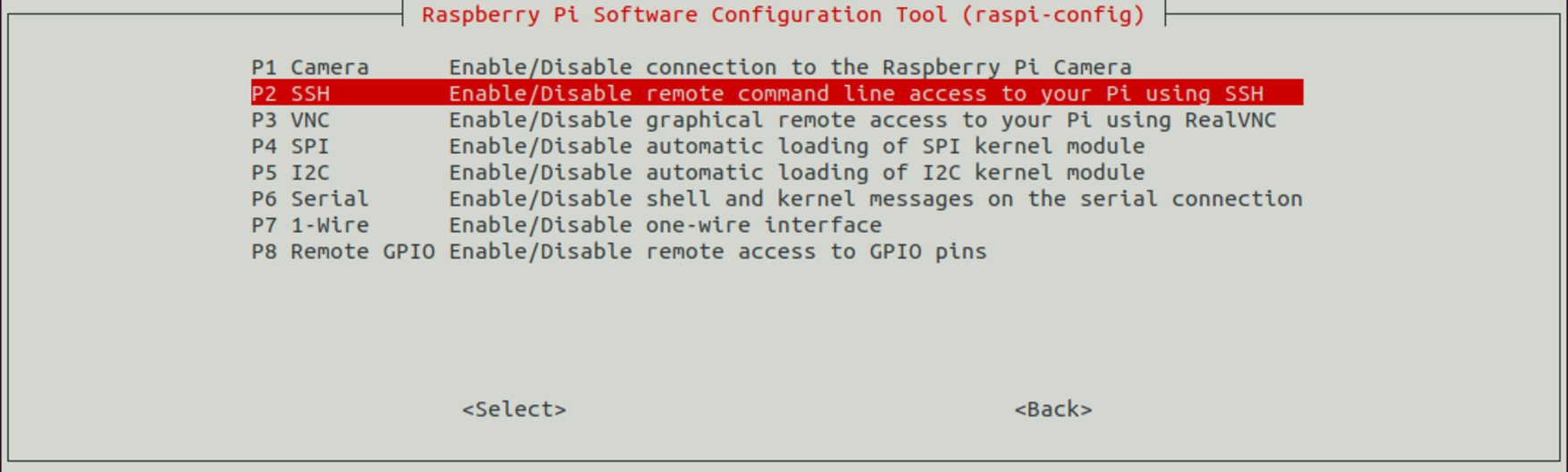
1. **On Raspberry Pi4:**

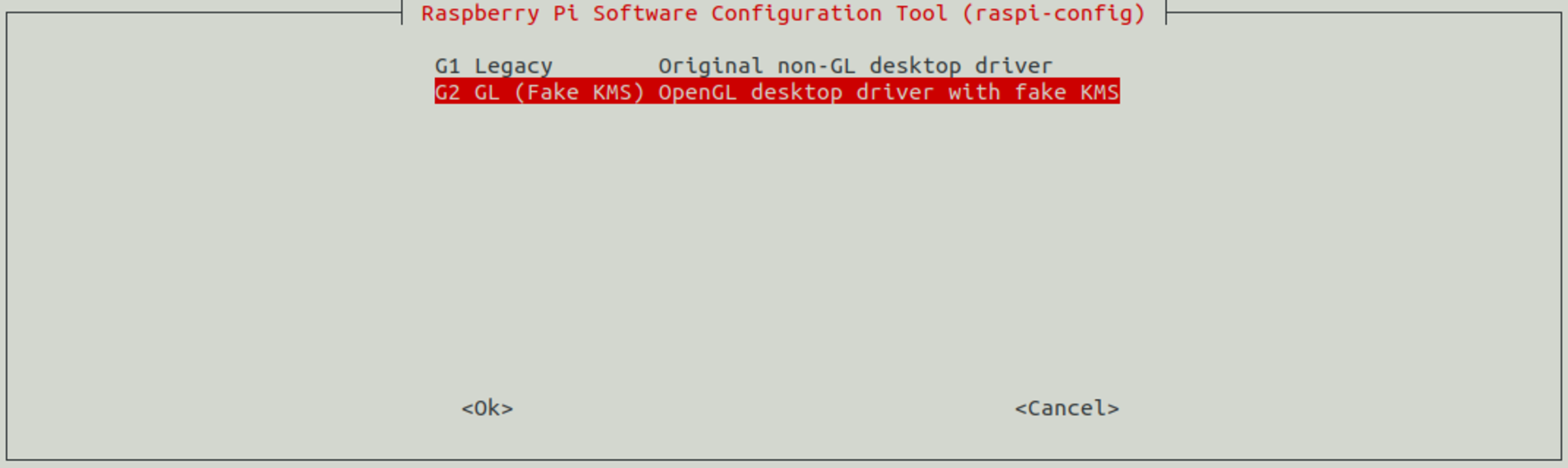
The steps shown below can also be performed on a "normal" Raspberry Pi 4.

When Pi 4 starts, open the configuration menu.

sudo raspi-config

"SSH" and "GL (Fake KMS)" are required for our configuration. See the two screenshots below.





**3.1** Then move the development resources to /etc/apt/sources.list. Add or uncomment the following line here: (open /etc/apt/sources.list file and uncoment below line)

deb-src http://raspbian.raspberrypi.org/raspbian/ buster main contrib non-free rpi

**3.2** Then update the system with the following commands

sudo apt update

sudo apt full-upgrade

sudo reboot

sudo rpi-update

sudo reboot

**3.3** Then install the required Qt and development packages

sudo apt-get build-dep qt5-qmake

sudo apt-get build-dep libqt5webengine-data

sudo apt-get install libboost1.58-all-dev libudev-dev libinput-dev libts-dev libmtdev-dev libjpeg-dev libfontconfig1-dev

sudo apt-get install libssl-dev libdbus-1-dev libglib2.0-dev libxkbcommon-dev libegl1-mesa-dev libgbm-dev libgles2-mesa-dev mesa-common-dev

sudo apt-get install libasound2-dev libpulse-dev gstreamer1.0-omx libgstreamer1.0-dev libgstreamer-plugins-base1.0-dev gstreamer1.0-alsa

sudo apt-get install libvpx-dev libsrtp0-dev libsnappy-dev libnss3-dev

sudo apt-get install "^libxcb.\*"

sudo apt-get install flex bison libxslt-dev ruby gperf libbz2-dev libcups2-dev libatkmm-1.6-dev libxi6 libxcomposite1

sudo apt-get install libfreetype6-dev libicu-dev libsqlite3-dev libxslt1-dev libavcodec-dev libavformat-dev libswscale-dev

sudo apt-get install libgstreamer0.10-dev gstreamer-tools libraspberrypi-dev libx11-dev libglib2.0-dev

sudo apt-get install freetds-dev libsqlite0-dev libpq-dev libiodbc2-dev firebird-dev libjpeg9-dev libgst-dev libxext-dev libxcb1 libxcb1-dev libx11-xcb1

sudo apt-get install libx11-xcb-dev libxcb-keysyms1 libxcb-keysyms1-dev libxcb-image0 libxcb-image0-dev libxcb-shm0 libxcb-shm0-dev libxcb-icccm4 libxcb-icccm4-dev

sudo apt-get install libxcb-sync1 libxcb-sync-dev libxcb-render-util0 libxcb-render-util0-dev libxcb-xfixes0-dev libxrender-dev libxcb-shape0-dev libxcb-randr0-dev

sudo apt-get install libxcb-glx0-dev libxi-dev libdrm-dev libssl-dev libxcb-xinerama0 libxcb-xinerama0-dev

sudo apt-get install libatspi-dev libssl-dev libxcursor-dev libxcomposite-dev libxdamage-dev libfontconfig1-dev

sudo apt-get install libxss-dev libxtst-dev libpci-dev libcap-dev libsrtp0-dev libxrandr-dev libnss3-dev libdirectfb-dev libaudio-dev

**3.4** Now create a directory for RaspberryQt

sudo mkdir /usr/local/qt5.15

sudo chown -R pi:pi /usr/local/qt5.15

1. **On Ubuntu Machine:**

**4.1** A computer or virtual machine with Ubuntu 20 LTS installed is required. First, update Ubuntu and install the necessary libraries:

sudo apt-get update

sudo apt-get upgrade

sudo apt-get install gcc git bison python gperf pkg-config gdb-multiarch

sudo apt install build-essential

**4.2** Creating a directory structure for raspberry libraries

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/build

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/tools

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/sysroot

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/sysroot/usr

sudo mkdir ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/sysroot/opt

sudo chown -R 1000:1000 ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4

cd ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4

**4.3 DOWNLOADING QT RESOURCES**

We download the Qt sources and open them in the raspberrypi4 directory:

sudo wget http://download.qt.io/archive/qt/5.15/5.15.2/single/qt-everywhere-src-5.15.2.tar.xz

sudo tar xfv qt-everywhere-src-5.15.2.tar.xz

Now we need to modify the mkspec file a bit to be able to use it with the compiler. For this,enter the following commands:

cp -R qt-everywhere-src-5.15.2/qtbase/mkspecs/linux-arm-gnueabi-g++ qt-everywhere-src- 5.15.2/qtbase/mkspecs/linux-arm-gnueabihf-g++

sed -i -e 's/arm-linux-gnueabi-/arm-linux-gnueabihf-/g' qt-everywhere-src- 5.15.2/qtbase/mkspecs/linux-arm-gnueabihf-g++/qmake.conf

1. **CROSS COMPILER DOWNLOAD**

I am using Linaro version 7.4.1 as the cross compiler. To do this, go to the tools directory and download and open the compiler:

cd tools

sudo wget https://releases.linaro.org/components/toolchain/binaries/7.4-2019.02/arm-linux-gnueabihf/gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf.tar.xz

tar xfv gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf.tar.xz

#### **REMOTELY MAPPING RASPBERRY LIBRARIES (RSYNC)**

Now, we need the original Raspberry Pi libraries that we copied to Ubuntu directories with rsync:put a ipconfig of ri-pi in below commands

cd ..

rsync -avz --rsync-path="sudo rsync" --delete pi@192.168.1.7:/lib sysroot/

rsync -avz --rsync-path="sudo rsync" --delete pi@192.168.1.7:/usr/include sysroot/usr/

rsync -avz --rsync-path="sudo rsync" --delete pi@192.168.1.7:/usr/lib sysroot/usr/

rsync -avz --rsync-path="sudo rsync" --delete pi@192.168.1.7:/opt/vc sysroot/opt/

Now we need to clear the symbolic links copied with rsync to point to the correct original files. A small Python script will be downloaded for this as well:

wget <https://raw.githubusercontent.com/riscv/riscv-poky/master/scripts/sysroot-relativelinks.py>

After downloading, set the necessary settings to run the script:

sudo chmod +x sysroot-relativelinks.py

./sysroot-relativelinks.py sysroot

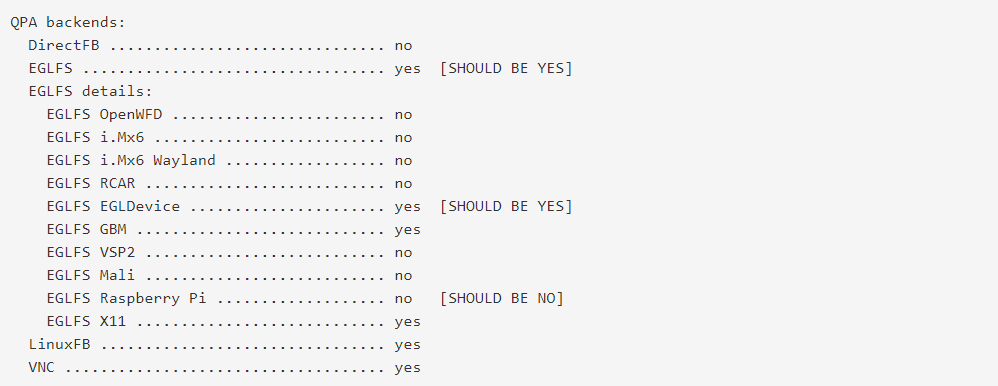
#### **QT COMPILATION**

Now we can configure and compile this version

cd build

../qt-everywhere-src-5.15.2/configure -release -opengl es2 -eglfs -device linux-rasp-pi4-v3d-g++ -device-option CROSS\_COMPILE=~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/tools/gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf/bin/arm-linux-gnueabihf- -sysroot ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/sysroot -prefix /usr/local/qt5.15 -extprefix ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/qt5.15 -opensource -confirm-license -skip qtscript -skip qtwayland -skip qtwebengine -nomake tests -nomake examples -make libs -pkg-config -no-use-gold-linker -v

The script ends after a few minutes and it is determined whether the following EGLFS conditions should be set.



If this does not happen or any other error message appears, please investigate the cause and correct the problem. If you want to rerun the configuration script with changed variables, be sure to delete the build directory content first. If everything is fine, execute the make and make install commands.

### **Build Qt:**

Our build has been configured now, and it is time to actually build the source files. Ensure you are still in the build directory, and run the following command:

make -j8

The -j4 option indicates that the job should be spread into 4 threads and run in parallel. I allocated 4 CPU cores to my Ubuntu virtual machine, so I would think the system will make use of that and distribute the workload among the 8 cores.

This process will take 4-5 hours.

Once it is completed, we can install the built package using the following command:

make install

This should install the files in the correct directories(~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/qt5.15)

### **Deploy Qt to our Raspberry Pi:**

We can now deploy Qt to our RPi. We will again make use of the rsync command. First move back into the rpi folder using the following command:

cd ..

You should now see a new folder named "qt5.15" here. Copy this to the raspberry pi using the following command [replace 192.168.1.7 with your RPi's IP address]:

rsync -avz --rsync-path="sudo rsync" qt5.15 [pi@192.168.1.7:/usr/local](mailto:pi@192.168.1.7:/usr/local)

## **Update linker on Raspberry Pi**

(I'm not entirely sure if this step is needed)

Enter the following command to update the device letting the linker to find the new Qt library files:

echo /usr/local/qt5.15/lib | sudo tee /etc/ld.so.conf.d/qt5.15.confsudo ldconfig

The Qt wiki for installing on a Raspberry Pi 2 suggests the following:

If you're facing issues with running the example, try to use 00-qt5pi.conf instead of qt5pi.conf, to introduce proper order.

Something to try if you're having issues running your projects.

That should be it! You have now (hopefully) succesfully installed Qt 5.15 on the Raspberry Pi 4B.

In the next step, we will build an example application, just to check everything works.

## **Build an example application**

On the PC, run the following commands to build one of the example OpenGL projects that comes bundled with Qt and deploy it to the Raspberry Pi:

To make a copy of the example project, run the following command:

cp -r ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/qt-everywhere-src-5.15.2/qtbase/examples/opengl/qopenglwidget ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/

Move into that new folder and build the source files with these commands:

cd ~/Documents/Qt-CrossCompile-RaspberryPi/raspberrypi4/qopenglwidget

../qt5.15/bin/qmake

make

Copy the built binary file to the Rasberry Pi with this command [change IP address to your RPi's]:

scp qopenglwidget [pi@192.168.1.7:/home/pi](mailto:pi@192.168.1.7:/home/pi)

Now **switch to the Raspberry Pi** and navigate to the home directory:

cd ~

Now run the compiled executable that we copied over from the host machine:

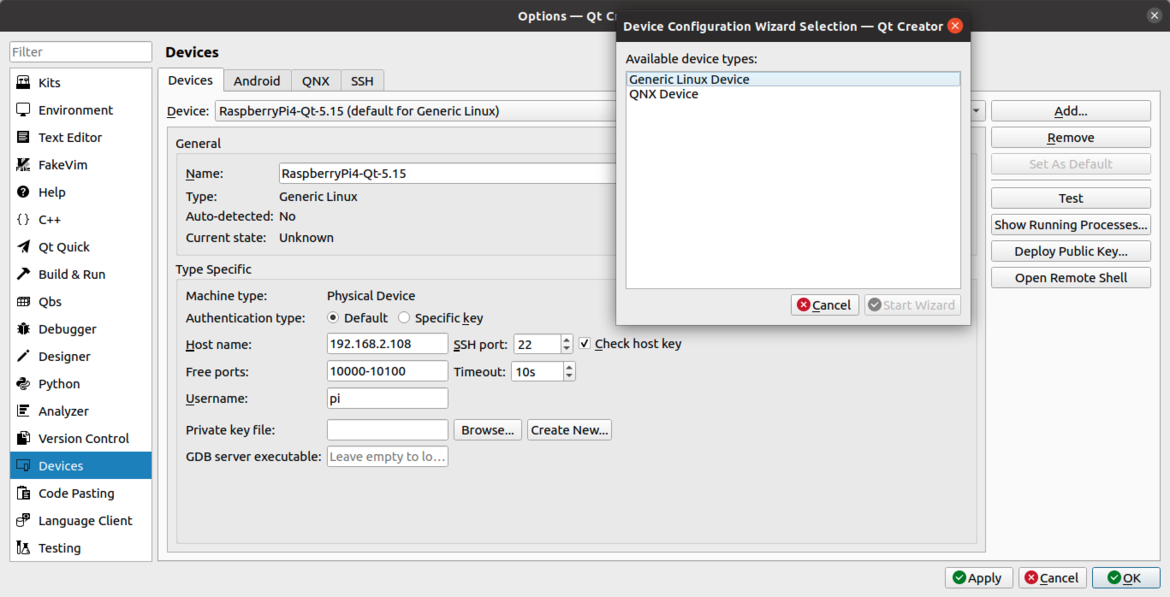
./qopenglwidget

The demo should start running on the display connected to the Raspberry Pi.

### **QT CREATOR CONFIGURATION:**

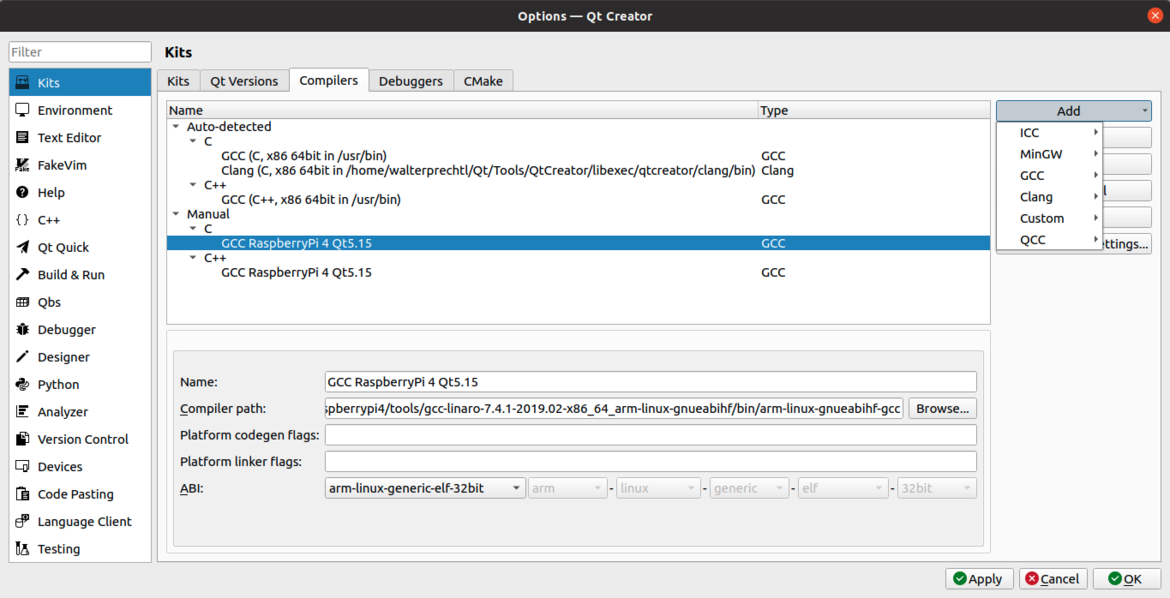
### **CREATE A DEVICE**

In the first step we create a new device. To do this, select "Options" from the menu under "Tools" and select "Devices" in the left column. Then use the "Add" function to create a new device called "Generic Linux Device". Name this device: Here the name RaspberryPi4-Qt-5.15 will be used. Enter the IP address under "Host name" and "pi" as "User name" as is typical for Raspberry. You can now test the Raspberry connection with the "Test" button. If the "Device test finished successfully" message does not appear here, you need to check the settings and see if there is actually access to the Raspberry with these parameters.



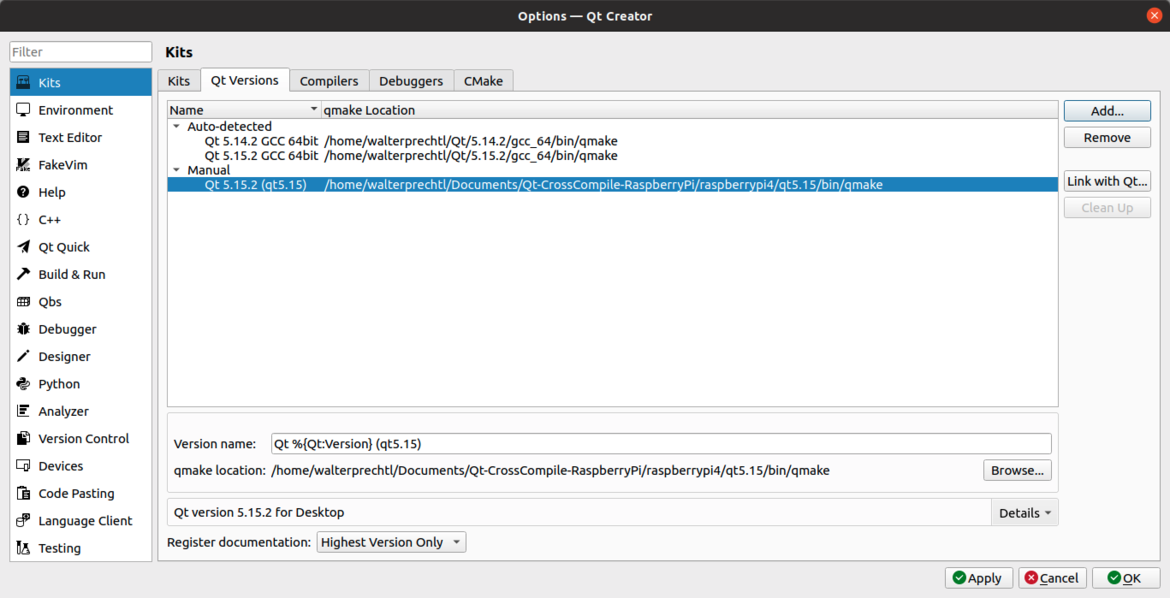
### **CONFIGURING THE COMPILER**

In the second step, we need to define a data path for C and C ++ compilers. You can find the related settings under "Kits -> Compilers" in the "Tools -> Options" menu again. Here, we are using the compilers we downloaded with the cross compiler "gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf". Two new configurations can be added by "Add -> GCC -> C" or "Add -> GCC -> C ++". One of them is "gcc-linaro-7.4.1-2019.02-x86\_64\_arm-linux-gnueabihf / bin / arm-linux-gnueabihf-gcc" in the "tools" directory for C and "gcc-linaro-7.4.1-2019.02-x86\_64\_arm" for C ++. Enter -linux-gnueabihf / bin / arm-linux-gnueabihf-g ++ ". This setting is ready after you have named each of them.



### **QT VERSIONING**

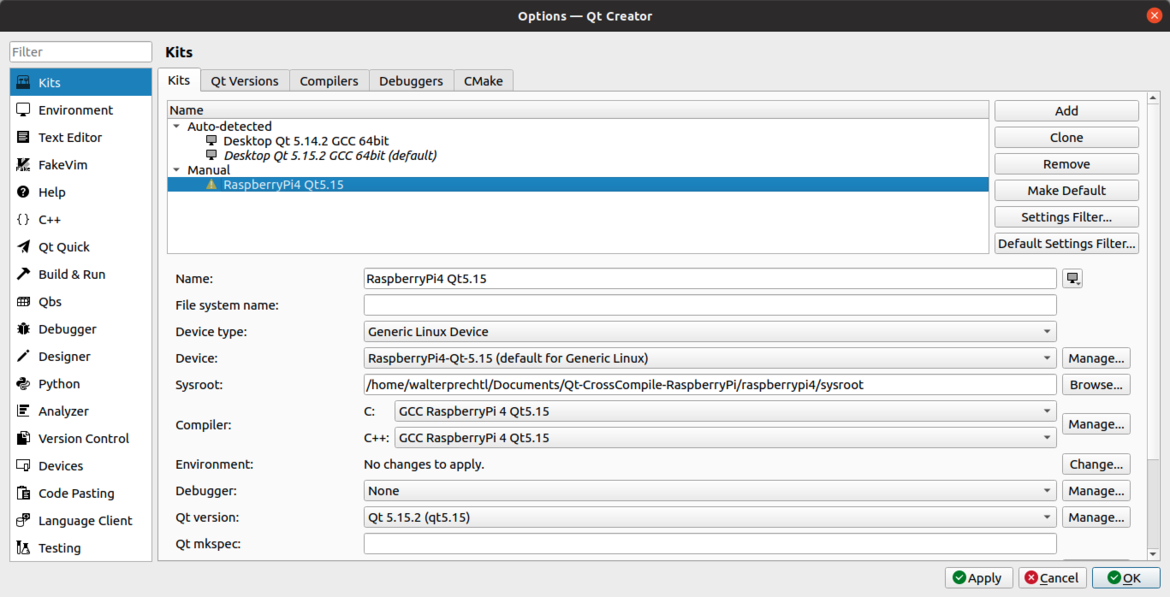
The third step requires a cross compiled qmake file from the previous blog post. Related settings can also be found under "Kits -> Qt Versions" in the "Tools -> Options" menu. Again add a new configuration with the "Add" function and select the qmake file from the "qt5.15 / bin / qmake" directory with the "Browse" button.



### **CREATING A KIT**

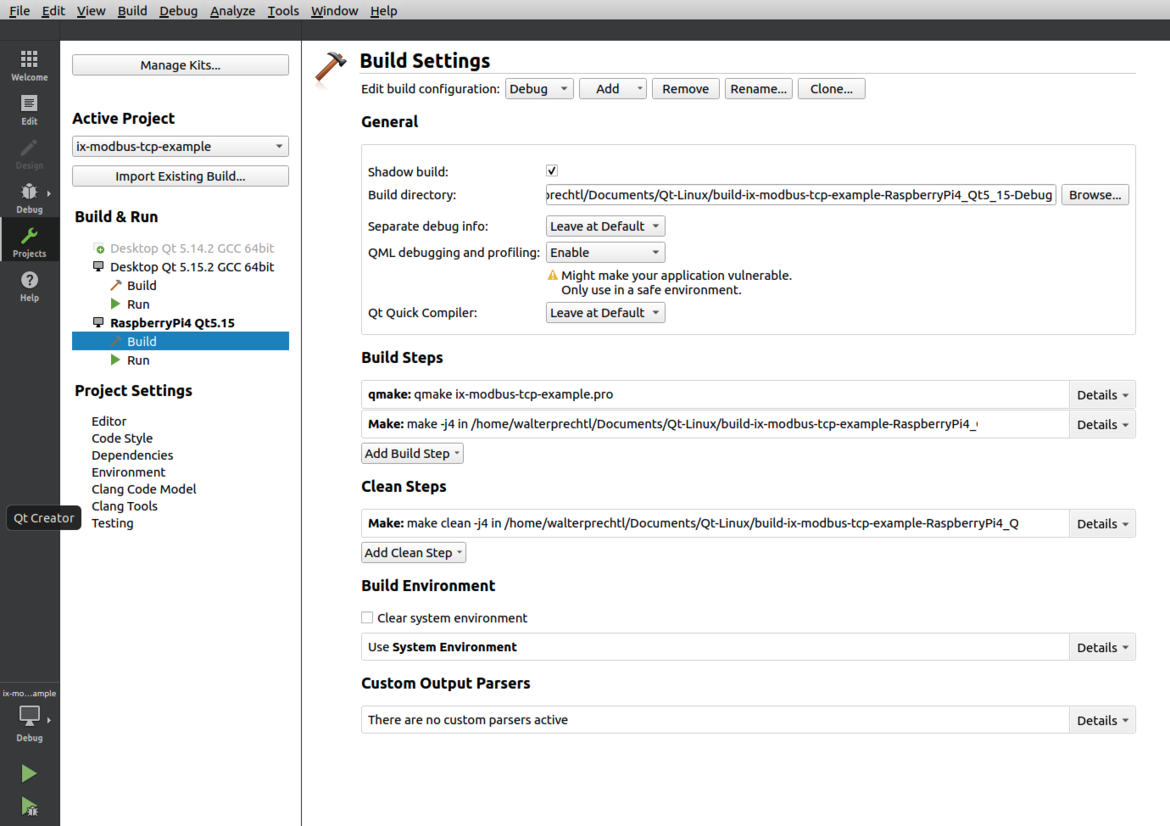
In the last step, a new kit is created by combining the newly added configurations. The settings for this are under "Kits -> Kits" in the "Tools -> Options" menu. Again add a new configuration with "Add" and set the following:

* Name: Give the configuration a "Name" (this name is used when you need to select the device from the "Project" settings later)
* Device type: Select "Generic Linux Device" as "Device type"
* Device: Select the newly created "Device"
* Sysroot: Select the sysroot directory described in the previous steps
* Compiler: Select both newly created "Compilers"
* Qt-Version: Select the newly created "Qt version"



### **PROJECT SETTINGS**

The newly created kit can now be selected immediately when creating a new project or added to an existing project.



To run applications from host to Raspberry pi display, we need to set some environment parameters in projects section.

Projects -> under **RaspberryPI4 Qt5.15** -> select **Run** -> In **Run Environment** section -> select **Batch Edit..** Button -> and paste below environment parameters

DISPLAY=:0.0

QT\_QPA\_PLATFORMTHEME=qt5ct

XAUTHORITY=/home/pi/.Xauthority

XDG\_SESSION\_TYPE=x11

Also in in .pro file, change the rules of deployment and change the target path to Raspi folder. It should look like below

qnx: target.path = /tmp/$${TARGET}/bin

else: unix:!android: target.path = /home/pi/$${TARGET}

!isEmpty(target.path): INSTALLS += target

Now you can run any application to Raspberry PI.

**Reference Links:**

* [**https://www.youtube.com/watch?v=TmtN3Rmx9Rk&t=2618s**](https://www.youtube.com/watch?v=TmtN3Rmx9Rk&t=2618s)
* [**https://www.interelectronix.com/de/qt-auf-dem-raspberry-pi-4.html**](https://www.interelectronix.com/de/qt-auf-dem-raspberry-pi-4.html)
* [**https://github.com/UvinduW/Cross-Compiling-Qt-for-Raspberry-Pi-4**](https://github.com/UvinduW/Cross-Compiling-Qt-for-Raspberry-Pi-4)