How to build a working Cross-Compile Environment for Wago PFC Devices

- **This guide was only tested on Ubuntu 19.04 !**

- Follow the steps in the Official Guide “https://github.com/WAGO/pfc-firmware-sdk” until you reach step 6.)

  → You should now have a working environment to build Applications for the Wago PFC.

- Get the latest(desired) Qt-Version.
  Either by downloading the Source directly from the Website:

  - “https://download.qt.io/archive/qt/5.13/5.13.0/single/”

  Or by downloading a specific Qt Version with git:

  - git clone -b 5.13.0 git://code.qt.io/qt/qt5.git
    Afterwards you need to call ./init_repository, optionally with the Modules you would like otherwise all modules will be downloaded.

  Info: For further information see: “https://wiki.qt.io/Get_the_Source”

- Copy the Folder “linux-arm-PFCXXXX-g++” into the downloaded Qt Folder:
  → qtbase → mkspecs → devices

- Open a Terminal in the basefolder of the qt-source. You should see a file named configure in the folder.

  Now we need to export some Variables for the Configuration, you should adjust the Paths to fit your paths.

  - “export TOOLCHAIN=*PATH_TO_YOUR_TOOLCHAIN*”
    For example the LINARO Toolchain used in the WAGO Tutorial
    → opt/wago/PFCXXXX/toolchain/arm-linux-gnueabihf/bin/arm-linux-gnueabihf-

  - “export HOST_SYSROOT=*PATH_TO_HOST_SYSROOT*”
    The host sysroot is the folder we created in the Wago Tutorial.
    You will find the Folder in ptxproj → platform-wago-pfcXXX → sysroot-host

  - “export TARGET_SYSROOT=*PATH_TO_TARGET_SYSROOT*”
    The Target sysroot is also the Folder we created in the Wago Tutorial
    You will find the Folder in ptxproj → platform-wago-pfcXXX → sysroot-target

  - “export PREFIX=*PATH_YOU_WANT_THE_QT_VERSION_TO_INSTALL_TO*
    The Path the newly build Qt-Version will install to.
    For example: /opt/Qt/5.13.0/wago_pfcxxx

  - “export EXTPREFIX=*PATH_YOU_WANT_THE_SYSROOT_TO_INSTALL_TO*
    By default the sysroot will be installed into the Toolchain folder. If you want the Sysroot

to install into a different folder you should set this Value, otherwise leave it out in the Configure.

- With all the Variables defined we can now build our Qt-Version for Cross-Compilation.

  - Execute the command:
    ./configure -device linux-arm-PFCXXXX-g++ -device-option CROSS_COMPILE=$TOOLCHAIN -device-option SYSROOT_TARGET=$TARGET_SYSROOT -sysroot $HOST_SYSROOT -prefix $PREFIX -extprefix=$EXTPREFIX -skip qtandroidextras -skip qtcharts -skip qtwinextras -skip qtlocation -skip qtwebengine -skip qtwebview -no-opengl -D WAGO_PFC

    Note: For further options see “https://doc.qt.io/archives/qtextended4.4/buildsystem/over-configure-options-1.html”

    → Follow the Prompts you will receive and check for a “success” message.

    - -device: The Device for which we will configure the build. Contains information about the Architecture and so on.

    - skip … : Skips the Qt-Module so it will not be build, you can enter all the Qt-Modules you do not want to include into your build.

    - No-opengl: The Wago-PFCXXX does not support opengl and it will not work either. So we need to tell Qt not to use it.

    - -D Add an explicit define to the build, so we can later determine our system in make/qmake

  - After the configure, run make -jn, n is the amount of threads your PC has + 1

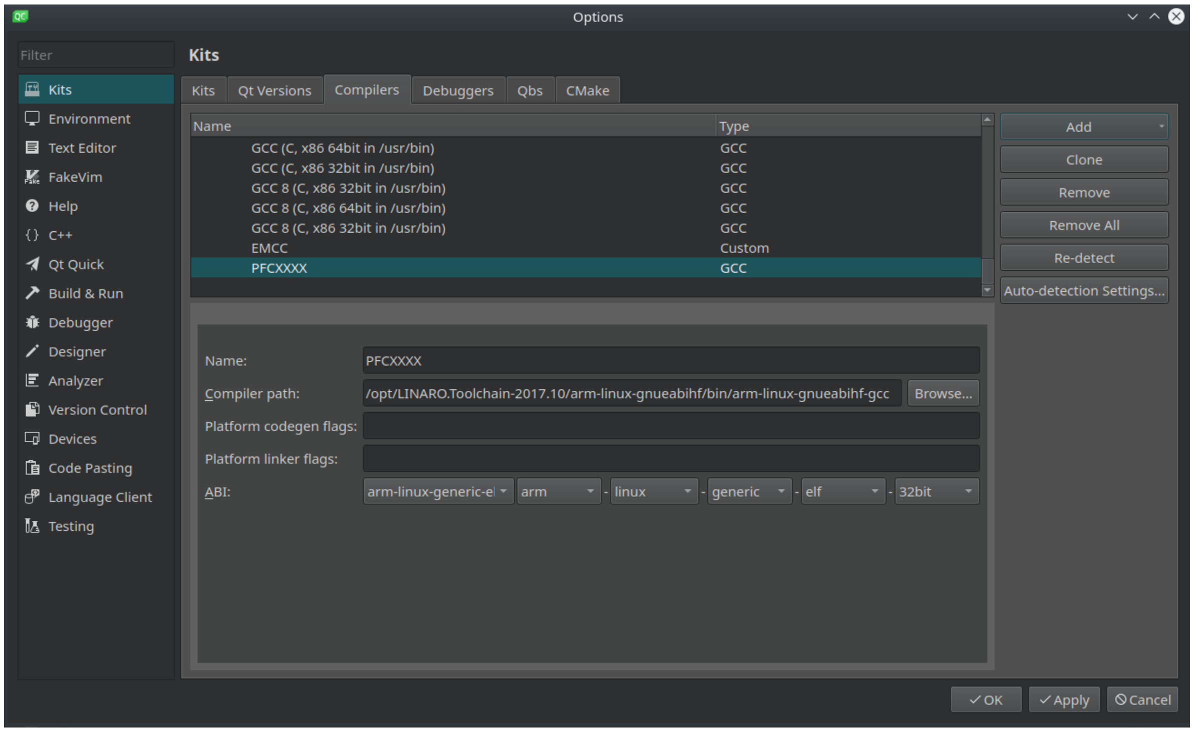
  - Afterwards the Qt-Version will be build, this will take some time even on faster machines, expect it to take around 30-90 minutes.

  - When the make is done, double check that no errors are listed in the last lines, if everything worked you are ready to install with: “make install”
    → This will install the build into the Path provided by $PREFIX

- That’s it we should now have a working Environment and can Cross Compile for the WagoPFC. If you need help configuring Qt to use the Build we just created, keep on reading.

<u>Setup QtCreator to use the Cross Compile Environment</u>

- If not already installed, download and install QtCreator → “https://www.qt.io/download”
- Open QtCreator. Navigate to Tools → Options → Kits
  - First setup the Compiler by clicking on the Tab Compilers:

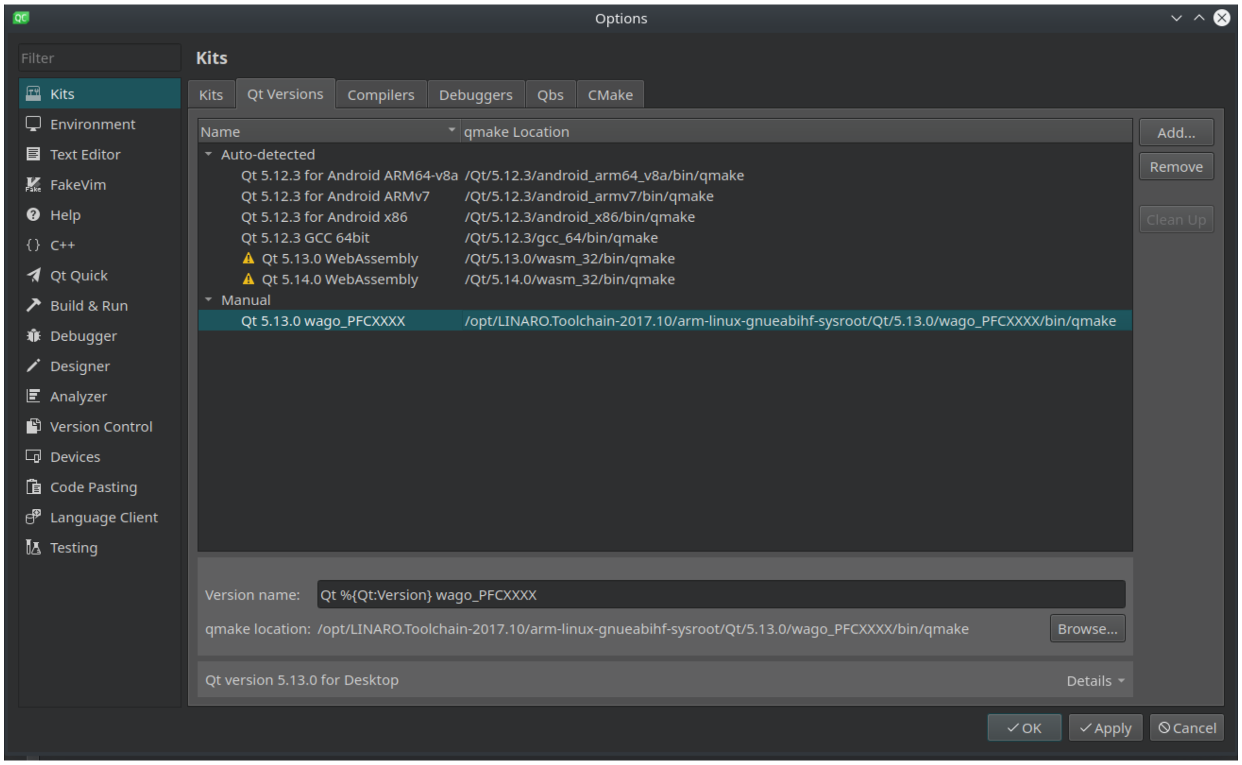


Name	Type
GCC (C, x86 64bit in /usr/bin)	GCC
GCC (C, x86 32bit in /usr/bin)	GCC
GCC 8 (C, x86 32bit in /usr/bin)	GCC
GCC 8 (C, x86 64bit in /usr/bin)	GCC
GCC 8 (C, x86 32bit in /usr/bin)	GCC
EMCC	Custom
PFCXXXX	GCC

Name:	PFCXXXX
Compiler path:	/opt/LINARO.Toolchain-2017.10/arm-linux-gnueabihf/bin/arm-linux-gnueabihf-gcc
Platform codegen flags:	
Platform linker flags:	
ABI:	arm-linux-generic-el arm - linux - generic - elf - 32bit

    - Click on Add → GCC → C and enter a name and compiler path as seen in the Picture above, the Toolchain should be the same as used in the configure above.
    - Repeat the step for the C++ Compiler: Add → GCC → C++, now use g++ instead of gcc

  - Configure the Debugger the same way we configured the Compilers but with the ending gdb.



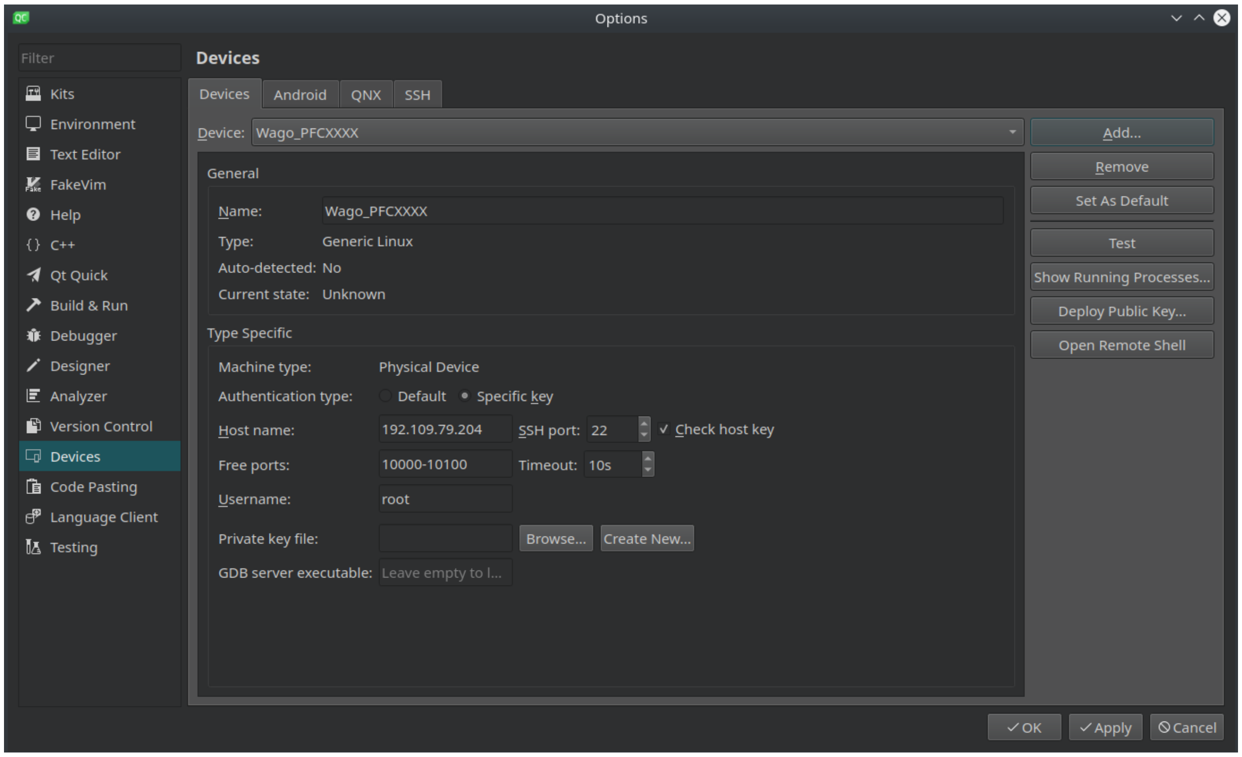
Name	qmake Location
Auto-detected	
Qt 5.12.3 for Android ARM64-v8a	/Qt/5.12.3/android_arm64_v8a/bin/qmake
Qt 5.12.3 for Android ARMv7	/Qt/5.12.3/android_armv7/bin/qmake
Qt 5.12.3 for Android x86	/Qt/5.12.3/android_x86/bin/qmake
Qt 5.12.3 GCC 64bit	/Qt/5.12.3/gcc_64/bin/qmake
⚠ Qt 5.13.0 WebAssembly	/Qt/5.13.0/wasm_32/bin/qmake
⚠ Qt 5.14.0 WebAssembly	/Qt/5.14.0/wasm_32/bin/qmake
Manual	
Qt 5.13.0 wago_PFCXXXX	/opt/LINARO.Toolchain-2017.10/arm-linux-gnueabihf-sysroot/Qt/5.13.0/wago_PFCXXXX/bin/qmake

Version name: Qt %{Qt:Version} wago_PFCXXXX

qmake location: /opt/LINARO.Toolchain-2017.10/arm-linux-gnueabihf-sysroot/Qt/5.13.0/wago_PFCXXXX/bin/qmake

Qt version 5.13.0 for Desktop

- ○ Now we need to add our Qt-Version:
  → If you added the Variable “EXTPREFIX” you will find the sysroot in that Path instead.
  Otherwise it will be in the Toolchain folder.



Device: Wago_PFCXXXX

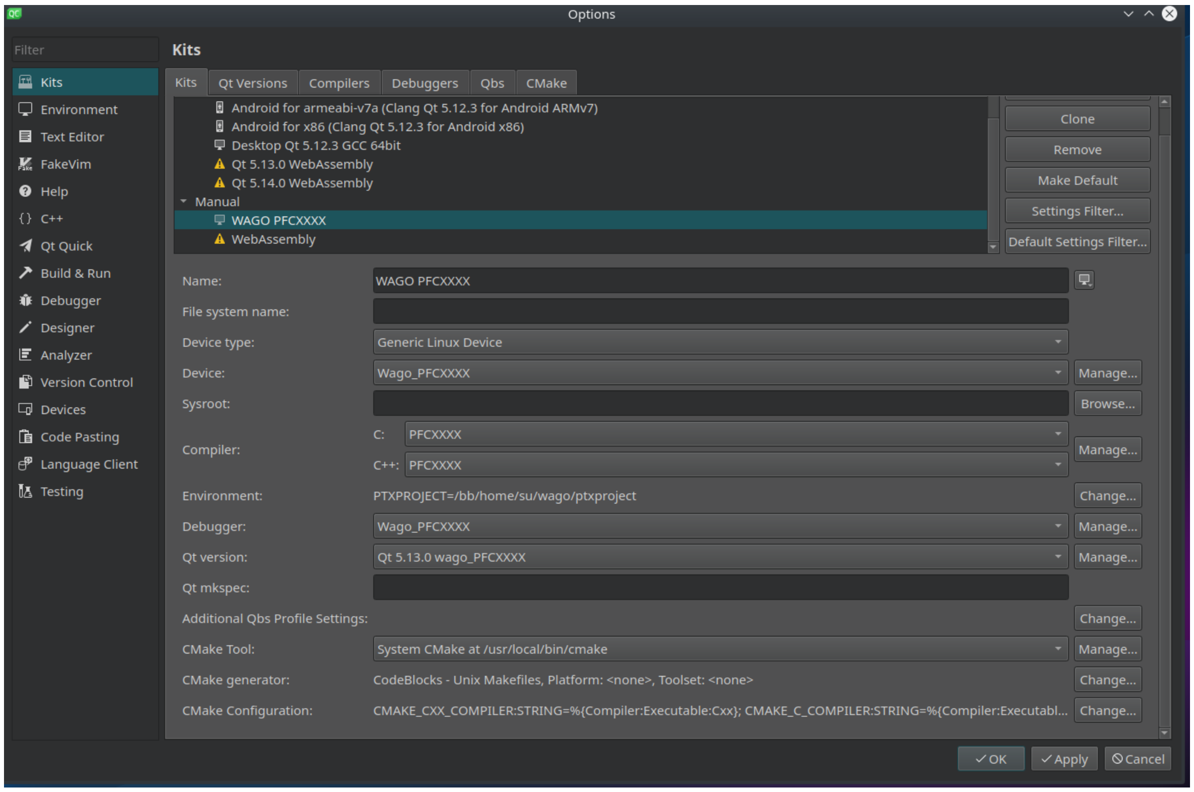
General

Name: Wago_PFCXXXX
Type: Generic Linux
Auto-detected: No
Current state: Unknown

Type Specific

Machine type: Physical Device
Authentication type: Default [✓ Specific key]
Host name: 192.109.79.204 SSH port: 22 ✓ Check host key
Free ports: 10000-10100 Timeout: 10s
Username: root
Private key file:
GDB server executable: Leave empty to l...

- ○ We should now add the Device to our Options, so we can remotely debug and deploy.
  Navigate to the Tab “Devices”.
  Click on Add → Generic Linux Device → Start Wizard and follow the prompts
  by entering the IP of your Wago PFC Device with the “sd.hdimg” Image.



Android for armeabi-v7a (Clang Qt 5.12.3 for Android ARMv7)
Android for x86 (Clang Qt 5.12.3 for Android x86)
Desktop Qt 5.12.3 GCC 64bit
Qt 5.13.0 WebAssembly
Qt 5.14.0 WebAssembly
Manual
WAGO PFCXXXX
WebAssembly

Name: WAGO PFCXXXX
File system name:
Device type: Generic Linux Device
Device: Wago_PFCXXXX
Sysroot:
Compiler: C: PFCXXXX C++: PFCXXXX
Environment: PTXPROJECT=/bb/home/su/wago/ptxproject
Debugger: Wago_PFCXXXX
Qt version: Qt 5.13.0 wago_PFCXXXX
Qt mkspec:
Additional Qbs Profile Settings:
CMake Tool: System CMake at /usr/local/bin/cmake
CMake generator: CodeBlocks - Unix Makefiles, Platform: <none>, Toolset: <none>
CMake Configuration: CMAKE_CXX_COMPILER:STRING=%{Compiler:Executable:Cxx}; CMAKE_C_COMPILER:STRING=%{Compiler:Executabl...

- ○ Finally we can add the Kit to Qt, navigate back to the Tab Kits:
  Click on Add and enter a name for your Kit:
  - ▪ Choose from the Device List our Wago PFC we just added. Make sure Device type is set to Generic Linux Device.
  - ▪ Add our C / C++ Compiler in the compiler section
  - ▪ Add our debugger in the Debugger section
  - ▪ Add the Qt Version in the Qt Version section.
- ○ We are finished now and can start programming for the Wago PFCXXX.

- Note: For an example, on how to work with Qt (qmake) see the Demo Program included.