**Qt5 cross compile for raspberry pi 3**

https://github.com/EliArad/MyDocuments

Pre-Requisites:

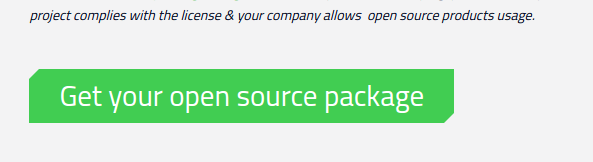
A. you need a raspberry pi board ( I used pi 3)

I used raspbian OS and not the outcome from buildroot, even though I used buildroot.

(Next steps will be to use the buildroot image and replace raspbian OS)

B. Download and extract buildroot from <https://buildroot.uclibc.org/>

C. download the open source QT from <https://info.qt.io/download-qt-for-application-development>



In order to make it working I managed to do the following

Using buildroot, I cross compile qt5

To allow QT5 we need to enable:

Depends on: BR2\_INSTALL\_LIBSTDCPP [=y] && BR2\_USE\_WCHAR [=y] && BR2\_TOOLCHAIN\_HAS\_THREADS\_NPTL [=y] && !BR2\_STATIC\_LIBS [=n] && !BR2\_PACKAGE\_QT [=n]

Quick notes on buildroot (I will create a different tutorial on what I know about buildroot)

I tried make those steps in Qt creator, but I did not success yes, because it did not found the makefile.

So this tutorial (which take me some time to understand all) will build the gui in QT Creator 5.8

But I will compile and copy the app to raspberry pi 3 using the command line.

I am using xconfig (the Qt gui version) but sometimes uses the ncurses menuconfig

(Dependencies will come up if they are missing, so just install them)

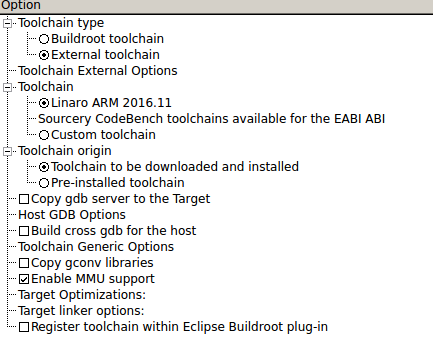
1. after downloading and extracting buildroot do:

make raspberrypi3\_defconfig

it will copy the raspberrypi3\_defconfig into .config in the local directory

2. Select external tool chain

I will save time in compile the tools chain

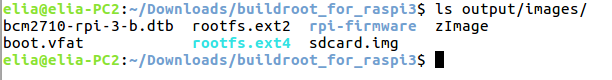


Once you have Qt5 , feel free to add everything you need or don’t :)



Once we have all of that, we can type make.

You should wait for two hours and then see that you have the images ready.



There are many tutorials in the internet which I got helped.

Many of them saying to download the img of the raspberry Linux and mount it for sysroot.

The idea is that we will use buildroot output and creation to cross compile QT.

The idea beyond qt cross compile is as follow:

If you go to output/host/usr where your top level build root is located you will be a directory called mkspecs



Every directory have two files:

qmake.conf qplatformdefs.h

Each file define where the compile will be taken from and which compiler for any of the necessary tools.

Gcc , g++, ar, and so on,

MAKEFILE\_GENERATOR = UNIX

CONFIG += incremental

QMAKE\_INCREMENTAL\_STYLE = sublib

include(../common/linux.conf)

include(../common/gcc-base-unix.conf)

include(../common/g++-unix.conf)

# modifications to g++.conf

QMAKE\_CC = arm-linux-gnueabi-gcc

QMAKE\_CXX = arm-linux-gnueabi-g++

QMAKE\_LINK = arm-linux-gnueabi-g++

QMAKE\_LINK\_SHLIB = arm-linux-gnueabi-g++

# modifications to linux.conf

QMAKE\_AR = arm-linux-gnueabi-ar cqs

QMAKE\_OBJCOPY = arm-linux-gnueabi-objcopy

QMAKE\_NM = arm-linux-gnueabi-nm -P

QMAKE\_STRIP = arm-linux-gnueabi-strip

load(qt\_config)

The first change we are going to do is to add a new mkspec:

We are using the hf gnu compiler

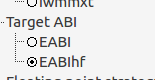
So let’s copy it from linux-arm-gnueabi-g++

cp -r linux-arm-gnueabi-g++/ linux-arm-gnueabihf-g++/

Remember, it is a directory of two files, so we do cp -r.

Now we need to edit the file and add the hf ( the hardware floating point compiler)

BTW, how do we know that we are using the EABIhf compiler?

From buildroot target architecture:

We only need to change the compiler to hf compile:

#

# qmake configuration for building with arm-linux-gnueabi-g++

#

MAKEFILE\_GENERATOR = UNIX

CONFIG += incremental

QMAKE\_INCREMENTAL\_STYLE = sublib

include(../common/linux.conf)

include(../common/gcc-base-unix.conf)

include(../common/g++-unix.conf)

# modifications to g++.conf

QMAKE\_CC = arm-linux-gnueabihf-gcc

QMAKE\_CXX = arm-linux-gnueabihf-g++

QMAKE\_LINK = arm-linux-gnueabihf-g++

QMAKE\_LINK\_SHLIB = arm-linux-gnueabihf-g++

# modifications to linux.conf

QMAKE\_AR = arm-linux-gnueabihf-ar cqs

QMAKE\_OBJCOPY = arm-linux-gnueabihf-objcopy

QMAKE\_NM = arm-linux-gnueabihf-nm -P

QMAKE\_STRIP = arm-linux-gnueabihf-strip

load(qt\_config)

I change every place where I have arm-linux-gnueabi-gcc to arm-linux-gnueabi**hf**-gcc

**Qt Creator**

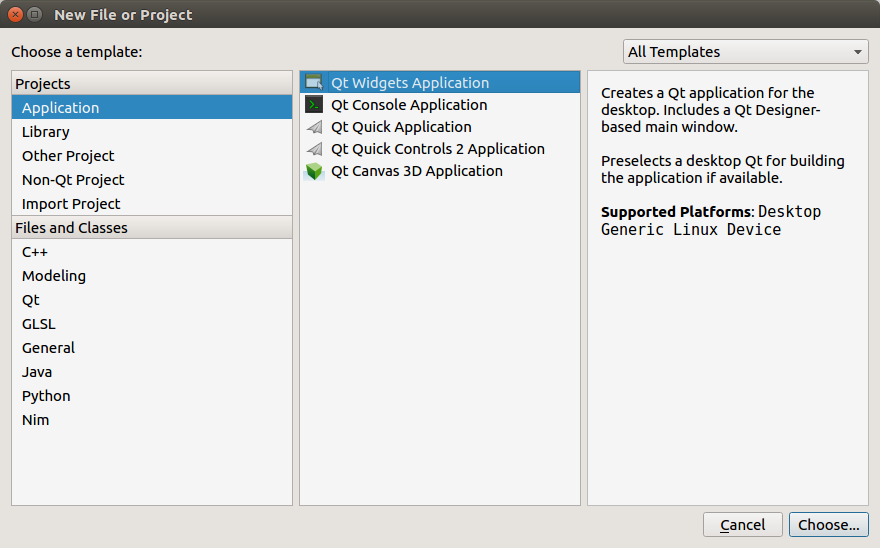
I did all the steps of creating device, declaring the compile and kit but it did not work

For now, which is also good step is to document and show how to build the gui in QT and compile in the command line.

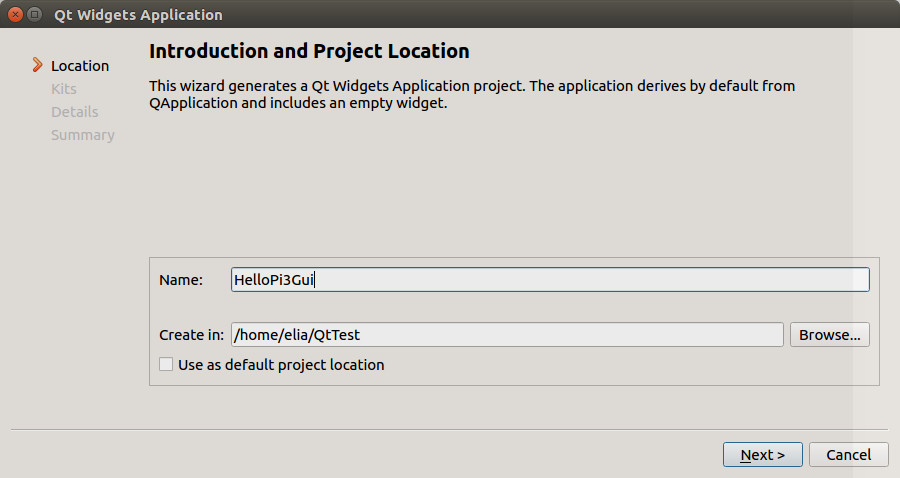
Steps:

1. Download QT Creator 5.X (I am using 5.8)

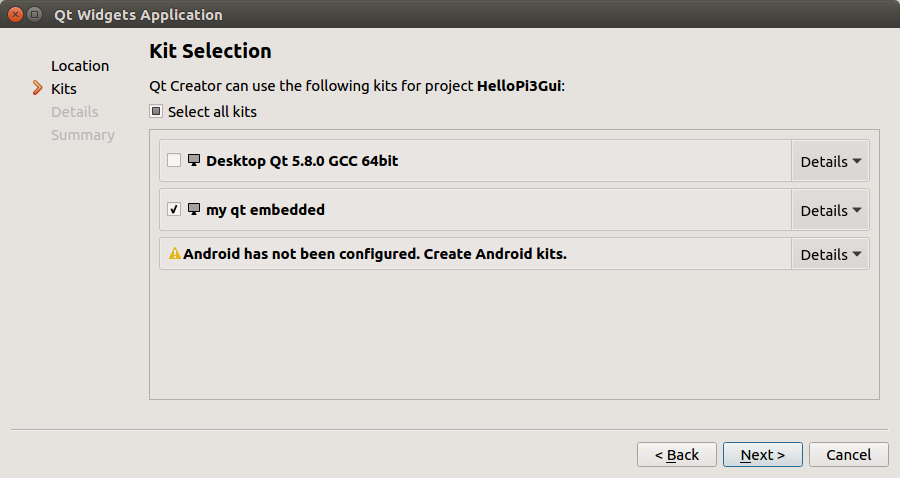
2. Create a GUI



Create the directory and the name of the GUI project



Select QT embedded

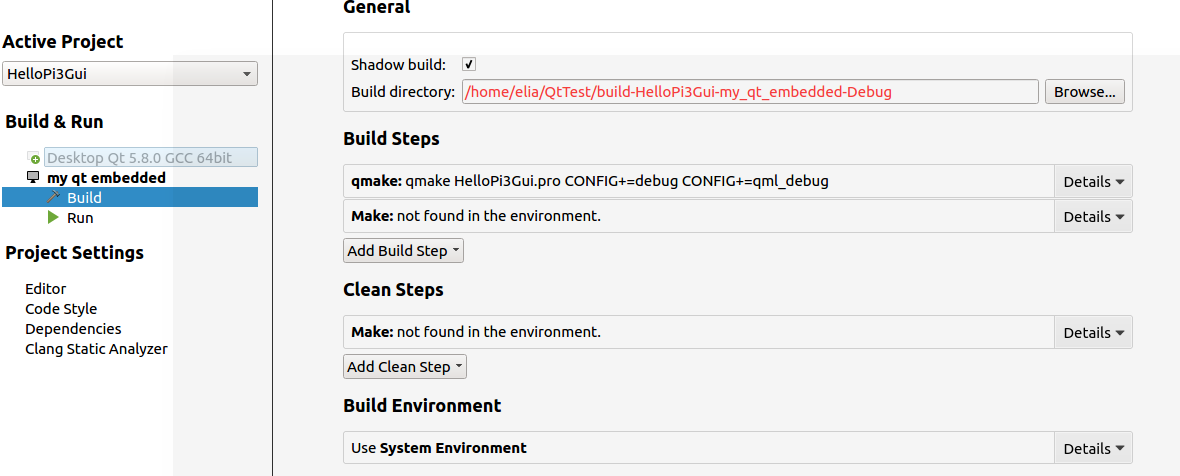
From did it came from?

I will show soon..

Then next, next and **finished**.

When I am going to the project tab:

I can see the make: not found in the environment,

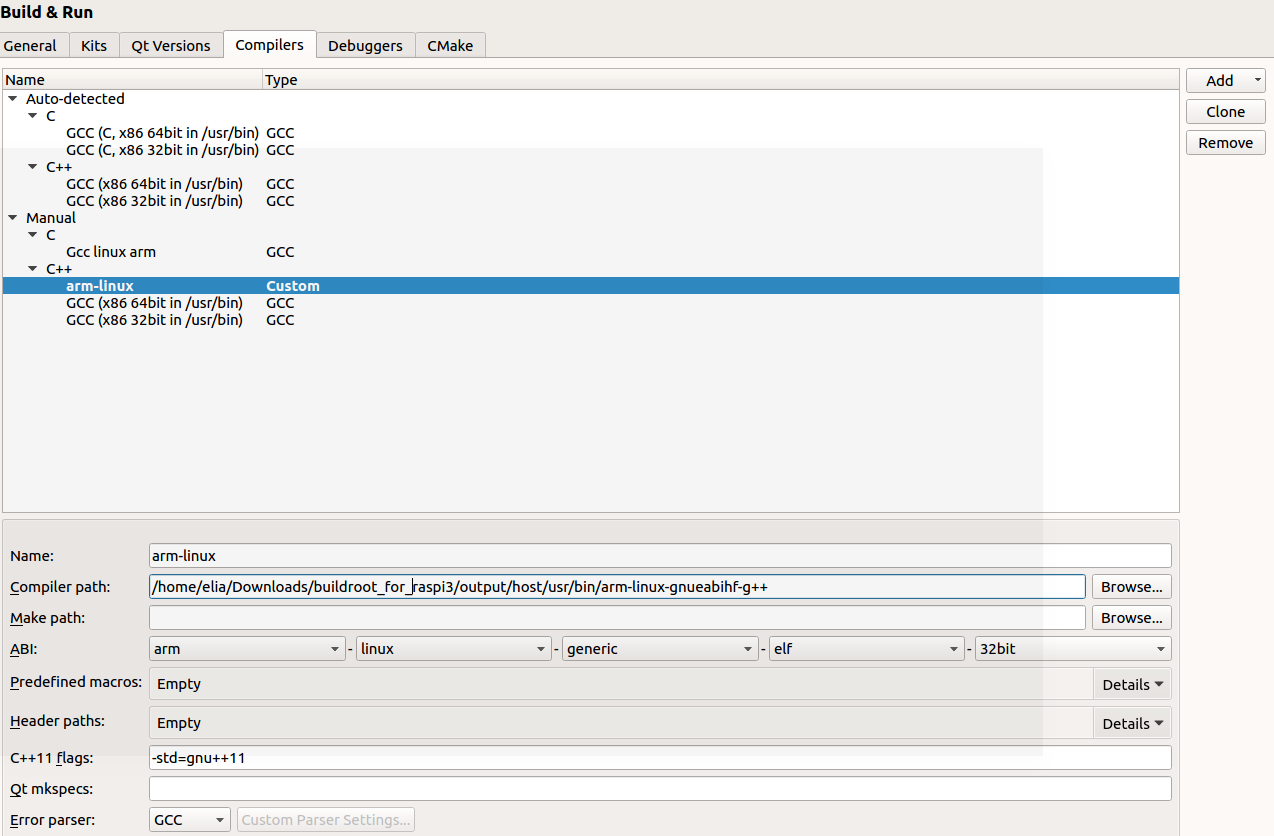


Before we continue to the command line compilation, lets see how did I create the device:

(For now it does not help much , but later it will be solved: )

*Go to tools→options*

In build and run, add a new compiler:



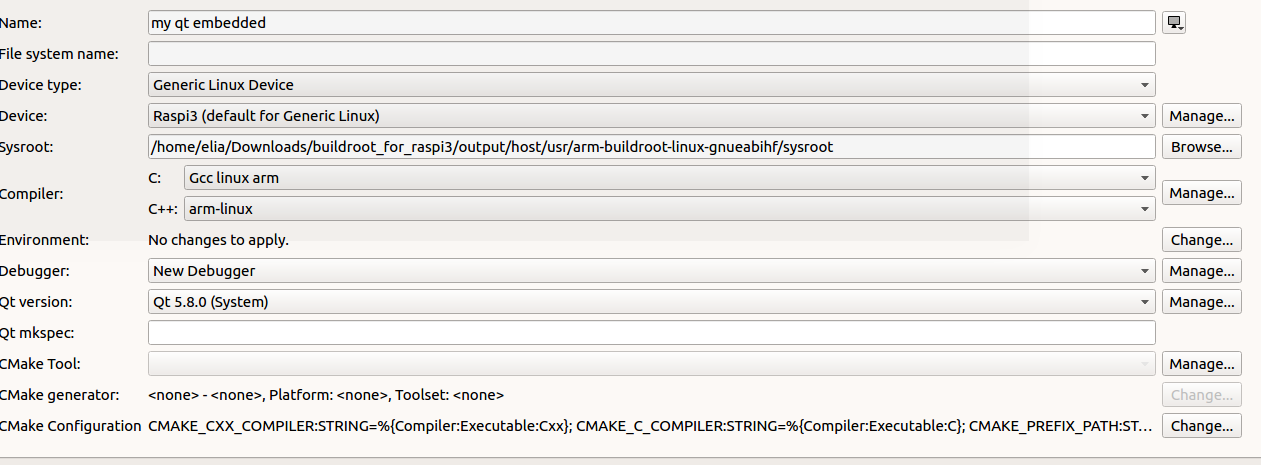
Name it as you want, I called it arm-Linux

In QT Versions, add a the qmake from buildroot:

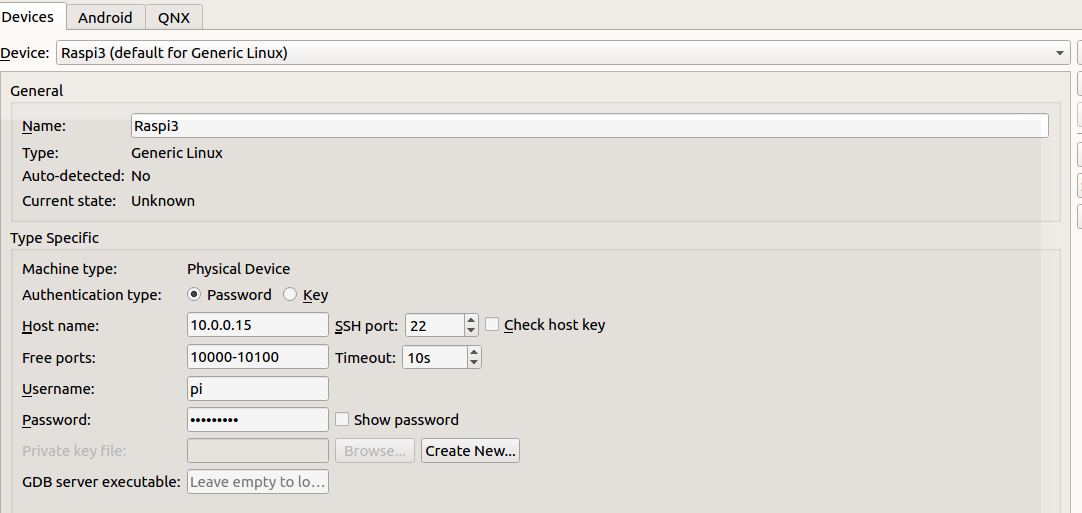


Go to Kits..

Add new..



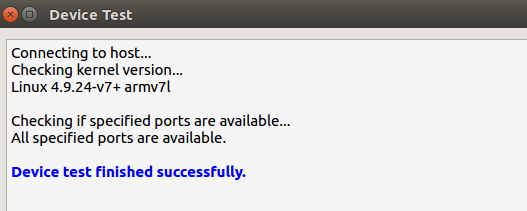
Add device type: (that you need also to create)



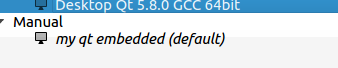
The user name for raspbian is pi and password is raspberry.

Enter name and IP address

You can do a test to see that the device is responding



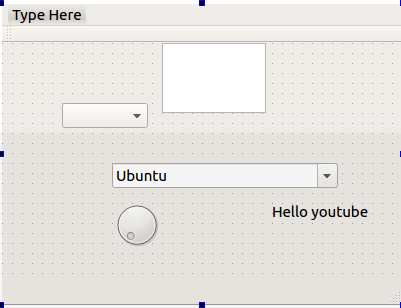
once your settings is ok , you will not see any red error:



Again, in this tutorial I choose to document my actions although I cannot compile directly from QT

open the edit mode and select the .ui file or just open the design mode

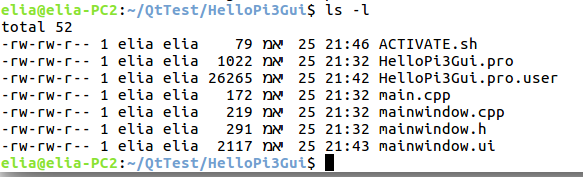
add some components.



**Now let’s go to compilation**

Open a terminal and move to the place where QT Creator creates your project

QtTest/HelloPi3Gui



I like to create a source script: name ACTIVATE.sh

1. add to the PATH the location of the build external toolchain,

In case you did not change the settings in buildroot, it will be located at output/host/usr/bin

export PATH=$PATH:/home/elia/Downloads/buildroot\_for\_raspi3/output/host/usr/bin

Create a **compile** script ( the killer line) or just run from the command line prompt:

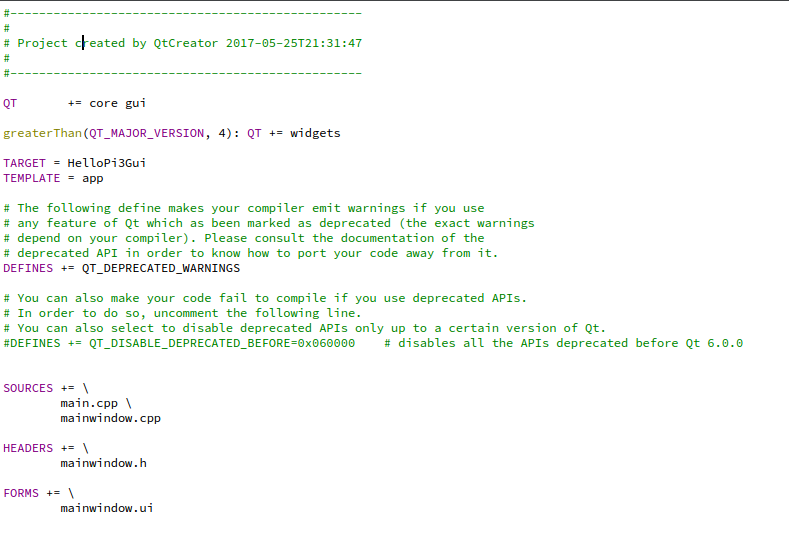
/home/elia/Downloads/buildroot\_for\_raspi3/output/host/usr/bin/qmake HelloPi3Gui.pro -spec linux-arm-gnueabihf-g++ CONFIG+=debug CONFIG+=qml\_debug

Let’s explain:

We are using the qmake from buildroot - see location

The HelloPi3Gui.pro is a file that was created by QTCreator

Go to the QTCreator project and you can see this file:



The compile script will run **qmake** to create a makefile for us,

This is what qmake is doing, its create a complicated makefiles.

After we run our compile script, we can see that we have **Makefile**

And what we do when we have Makefile - we call to make.

If we will now run the make we will get many errors.

./../Downloads/buildroot\_for\_raspi3/output/host/usr/arm-buildroot-linux-gnueabihf/sysroot/usr/include/qt5/QtCore/qbasicatomic.h:61:4: error: #error "Qt requires C++11 support" # error "Qt requires C++11 support"

Here I did the following:

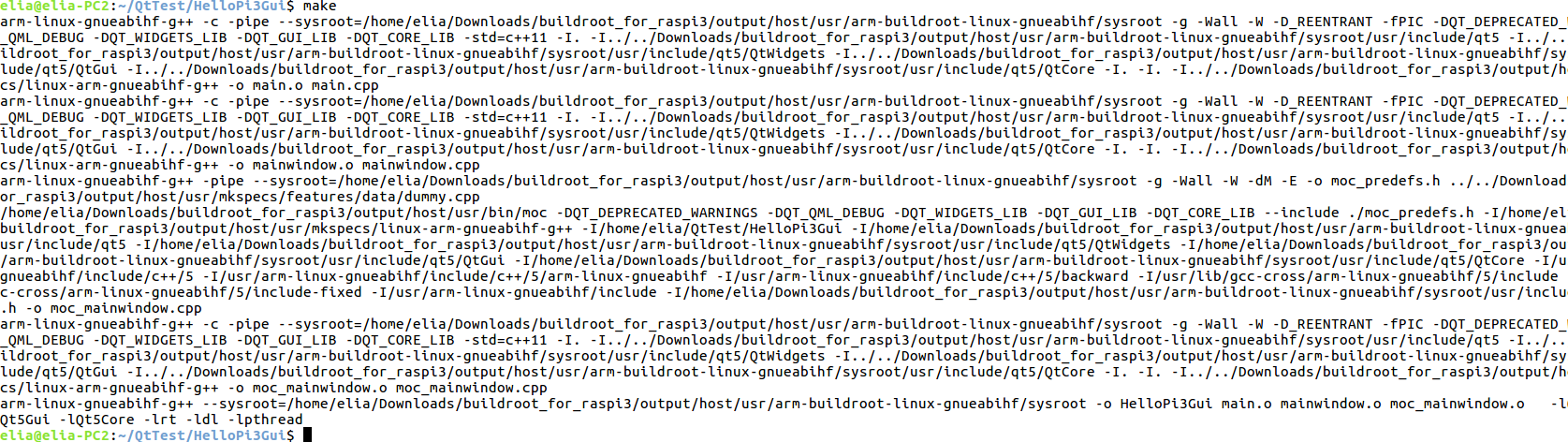
I edit (using joe editor) the Makefile

And added c++ 11 support:

CXXFLAGS = -pipe --sysroot=/home/elia/Downloads/buildroot\_for\_raspi3/output/host/usr/arm-buildroot-linux-gnueabihf/sysroot -g -Wall -W -D\_REENTRANT -fPIC $(DEFINES) -std=c++11

For CXXFLAGS

And now the make ended without any errors



Do file HelloPi3Gui shows that this file is ARM. (To see the file type and more info)

~/QtTest/HelloPi3Gui$ file HelloPi3Gui

HelloPi3Gui: ELF 32-bit LSB executable, ARM, EABI5 version 1 (GNU/Linux), dynamically linked, interpreter /lib/ld-linux-armhf.so.3, for GNU/Linux 3.2.0, BuildID[sha1]=7eba50bd1d4484bbc74dce9b3c7f16730de093ca, not stripped

That’s it, the file is ready to be copy to raspberry pi 3 using ssh:

sudo scp HelloPi3Gui [pi@10.0.0.15](mailto:pi@10.0.0.15):/home/pi

If you want to copy without the password every time:

sudo apt-get install sshpass

sudo sshpass -p "raspberry" scp HelloPi3Gui pi@10.0.0.15:/home/pi

Let’s summery the steps we made:

1. Download buildroot

2. Make def config for raspberry pi 3

3. Add QT5, after adding the dependencies ( if not it will not show in the menu)

4. Compile buildroot – wait for two hours until it needed.

5. Install QT Creator 5.8

6. We did define the QT embedded kit , although we did not compiled from QT Creator it self

7. We run qmake (the buildroot qmake ) with arguments of mkspec hf compiler and our .pro file

8. We got a Makefile, edit the Makefile to add C++ 11 flag support

9. We run make to produce the binary executable for our application.

10. Copy using SCP the binary to raspberry pi 3.

11. I run it there.

I wanted to use GUI and not console because I tried the similar steps with console app and QT4 (the default of buildroot)

There I were able to compile from the QT Creator and deploy to the target.

But when I tried to build a widget gui it failed to start.

I read somewhere the QT4 does not work so I moved to QT5 and it does run.

More notes:

Because I am using raspbian OS I needed to install QT requirements on the target:

We have INTERNET and apt-get so we can install qt Platform Plugin Dependencies

http://doc.qt.io/qt-5/linux-requirements.html

And we can find many of them in the web.

Upcoming documents:

\* Top level of understanding build root, create img deploy and test

\* Try fix the above issue and use QT Creator project full to compile and deploy to targeted

\* do the same for beagle bone black

\* Do the same for Yocto project.