**The following procedure is tested on Ubuntu Linux 16.04.2 LTS**

**Installing yocto toolchain**

Please download the file “fslc-framebuffer.tar.gz” and extract the file to “/opt” folder using below command

#sudo tar -xvzf fslc-framebuffer.tar.gz -C /opt/

# Configuring Qt Creator for Development

The examples here were done on Ubuntu Linux 16.04.2 LTS. The process would be similar on other desktop Linux distributions.

The assumption is that you already followed the above procedure( Installing yocto toolchain) to install toolchain/SDK with support for Qt.

I also assume you have a working Qt Creator, probably already set up for a desktop version of Qt. The steps and screenshots in this post will be based on the the most recent version of Qt Creator at the time of writing **Qt Creator 4.3.1**

## Configuration

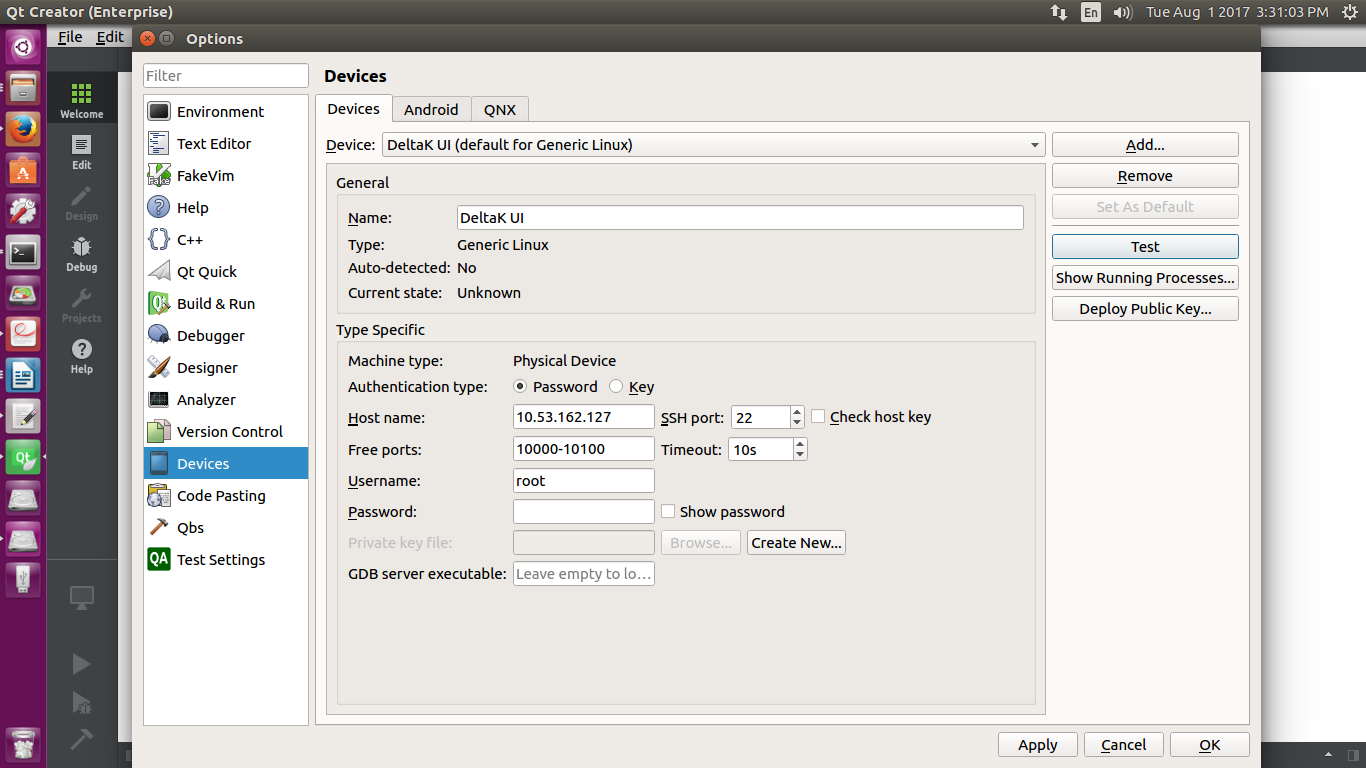
To set up Qt Creator for our Yocto toolchain, we need to do the following:

1. Set up the device.
2. Add the cross-compiler.
3. Add the debugger.
4. Add the Qt version.
5. Add a kit.
6. Configure the project.

## **Set Up the Device**

We need to set up our embedded system as a device. In my case it will be the **DeltaK-UI.** To do this Select Tools/Options... and Click on the Devices tab. Click Add..., select Generic Linux Device then Start Wizard. Give the device a name, enter the hostname or network IP address, and login credentials. Select Next and then Finish.

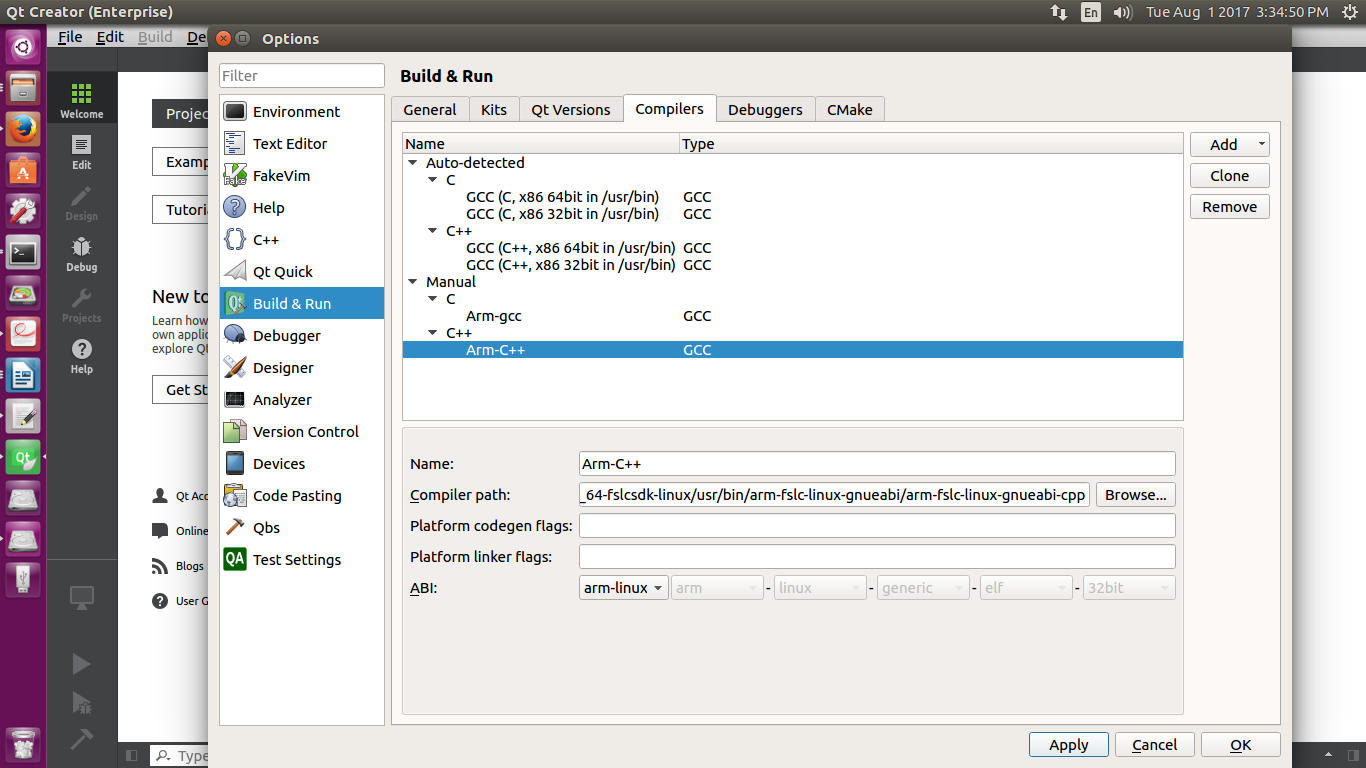
Qt Creator will test connectivity with the device, so you should have it up and running when you do this, if possible. The settings I used for the **DeltaK-UI** are shown below:



## **Add the Cross-Compiler**

Now we need to add our cross-compiler. In the Tools/Options screen, click on the Build & Run tab. Then click on the Compilers tab.

You'll probably see one or more compilers already there for native development. We add the Yocto cross-compiler by clicking on Add / GCC / C++. Give it a suitable name and enter the path to the cross-compiler. For the Yocto SDK I was using, the settings are shown below:



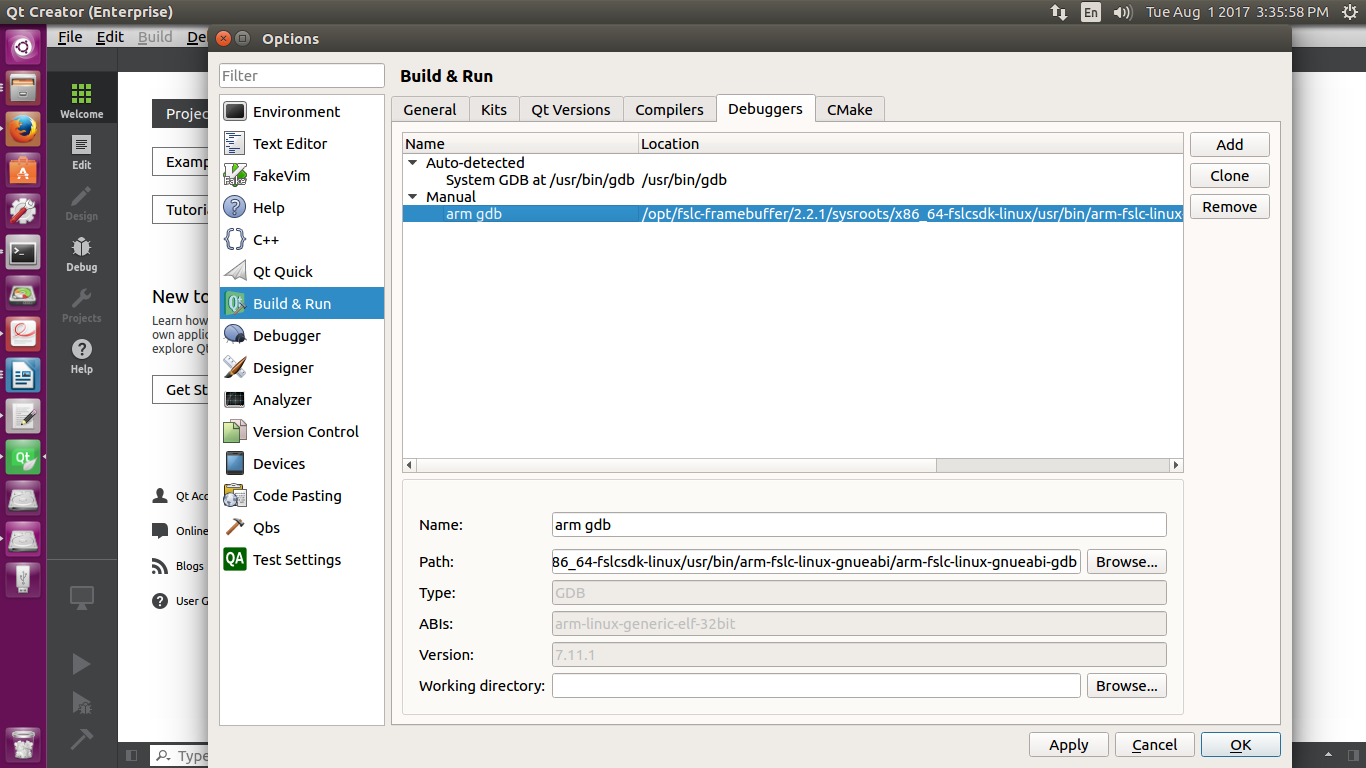
Select the

C compiler path to “/opt/fslc-framebuffer/2.2.1/sysroots/x86\_64-fslcsdk-linux/usr/bin/arm-fslc-linux-gnueabi/arm-fslc-linux-gnueabi-gcc”

and C++ compiler path to “/opt/fslc-framebuffer/2.2.1/sysroots/x86\_64-fslcsdk-linux/usr/bin/arm-fslc-linux-gnueabi/arm-fslc-linux-gnueabi-cpp”

## Add the Debugger

I won't cover debugging and other tools in this blog post, but in order to debug we need to add a suitable debugger. Click on the Debuggers tab, click on Add, and enter the name and path to your cross-debugger. The settings I used are shown below:

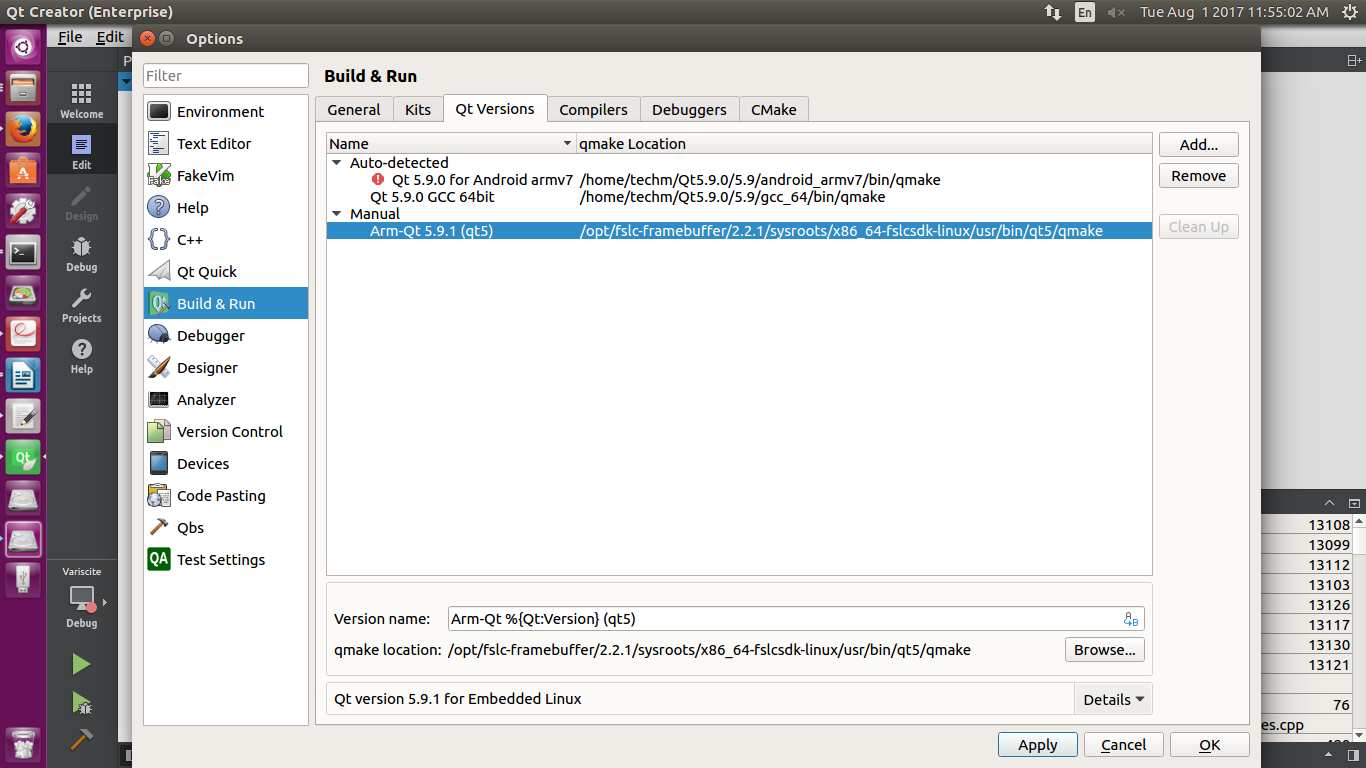
****

Select the

gdb path to “/opt/fslc-framebuffer/2.2.1/sysroots/x86\_64-fslcsdk-linux/usr/bin/arm-fslc-linux-gnueabi/arm-fslc-linux-gnueabi-gdb”

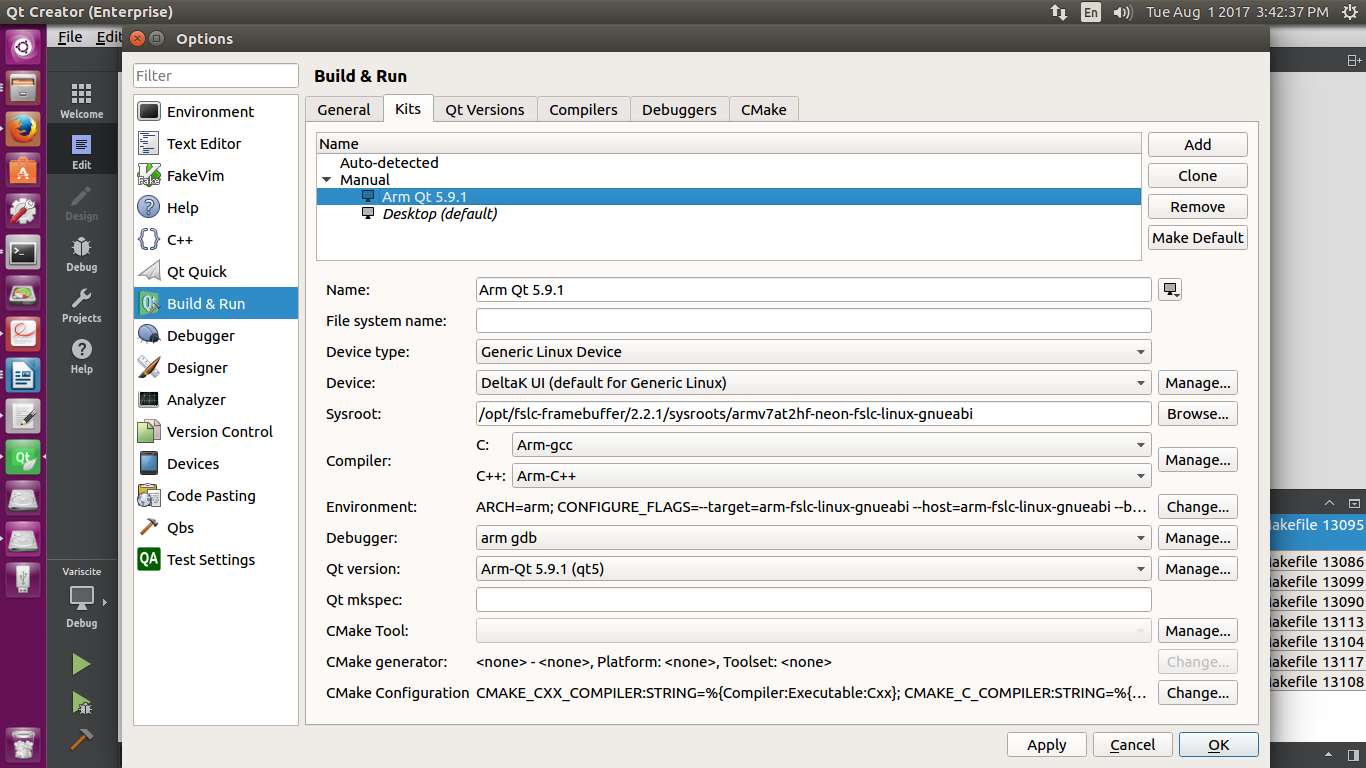
## **Add the Qt Version**

Next we add the Qt version supplied by the Yocto SDK. Click on the Qt Versions tab and then Add... Navigate to the qmake binary from your SDK. In my case it was “/opt/fslc-framebuffer/2.2.1/sysroots/x86\_64-fslcsdk-linux/usr/bin/qt5/qmake”, as shown below:



## **Configure a Kit**

The last step is to configure a "kit" which brings together a device, compiler, debugger, and Qt version. Click on the Kits tab and then Add. Pick a suitable name and select the device type and device. Enter the sysroot for your toolchain, the C++ and C compilers you defined earlier, as well as the debugger and Qt version. My kit settings are shown below:

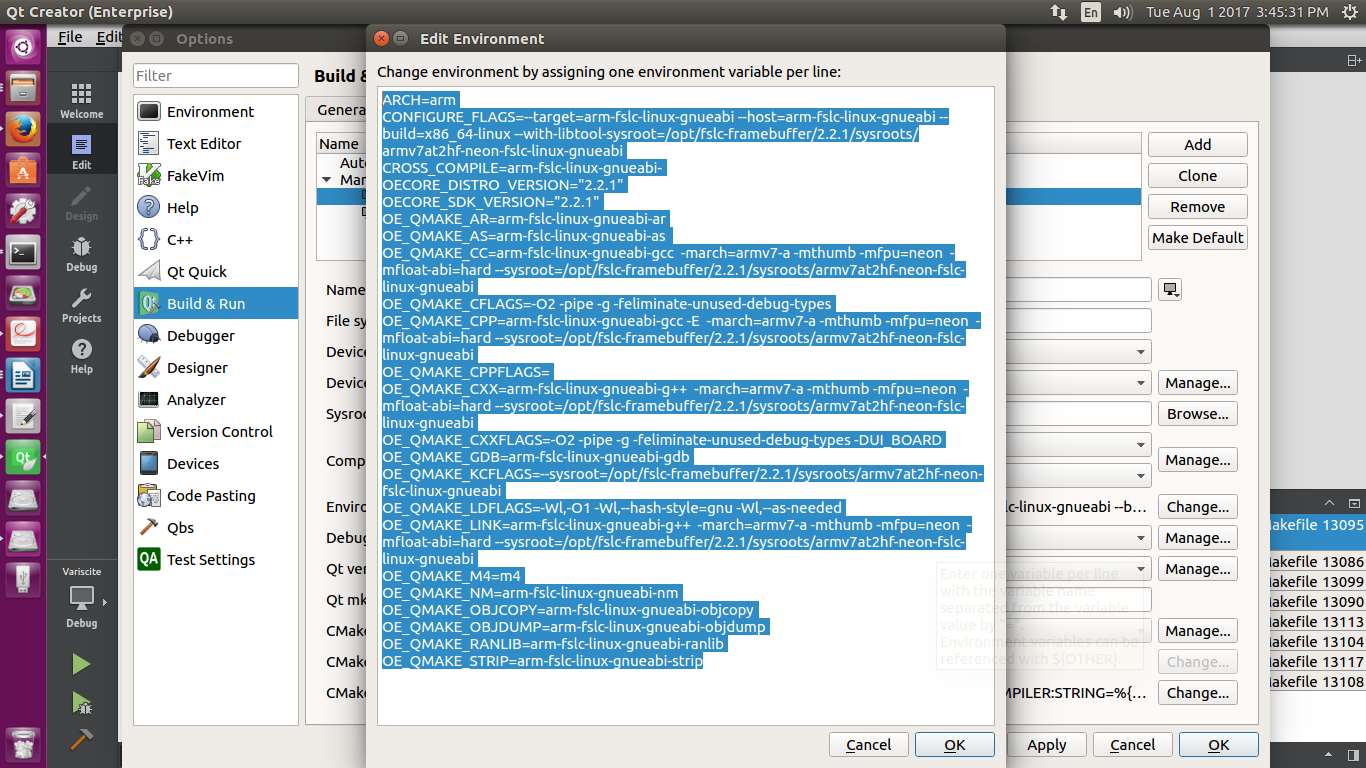
****

## Set Up the Environment

We need to do one more thing. We need to set up the environment using Qt Creator. We can set this up following way:

1. You can add the environment variables to the kit’s settings.

In my case I determined the environment needed by running the environment script and seeing what variables it set. You can then click on Change... under Environment: for the kit and paste in the environment variables, one per line. This is what I used:



Here are the actual settings in case you want to copy and paste them:

ARCH=arm

CONFIGURE\_FLAGS=--target=arm-fslc-linux-gnueabi --host=arm-fslc-linux-gnueabi --build=x86\_64-linux --with-libtool-sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

CROSS\_COMPILE=arm-fslc-linux-gnueabi-

OECORE\_DISTRO\_VERSION="2.2.1"

OECORE\_SDK\_VERSION="2.2.1"

OE\_QMAKE\_AR=arm-fslc-linux-gnueabi-ar

OE\_QMAKE\_AS=arm-fslc-linux-gnueabi-as

OE\_QMAKE\_CC=arm-fslc-linux-gnueabi-gcc -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

OE\_QMAKE\_CFLAGS=-O2 -pipe -g -feliminate-unused-debug-types

OE\_QMAKE\_CPP=arm-fslc-linux-gnueabi-gcc -E -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

OE\_QMAKE\_CPPFLAGS=

OE\_QMAKE\_CXX=arm-fslc-linux-gnueabi-g++ -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

OE\_QMAKE\_CXXFLAGS=-O2 -pipe -g -feliminate-unused-debug-types -DUI\_BOARD

OE\_QMAKE\_GDB=arm-fslc-linux-gnueabi-gdb

OE\_QMAKE\_KCFLAGS=--sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

OE\_QMAKE\_LDFLAGS=-Wl,-O1 -Wl,--hash-style=gnu -Wl,--as-needed

OE\_QMAKE\_LINK=arm-fslc-linux-gnueabi-g++ -march=armv7-a -mthumb -mfpu=neon -mfloat-abi=hard --sysroot=/opt/fslc-framebuffer/2.2.1/sysroots/armv7at2hf-neon-fslc-linux-gnueabi

OE\_QMAKE\_M4=m4

OE\_QMAKE\_NM=arm-fslc-linux-gnueabi-nm

OE\_QMAKE\_OBJCOPY=arm-fslc-linux-gnueabi-objcopy

OE\_QMAKE\_OBJDUMP=arm-fslc-linux-gnueabi-objdump

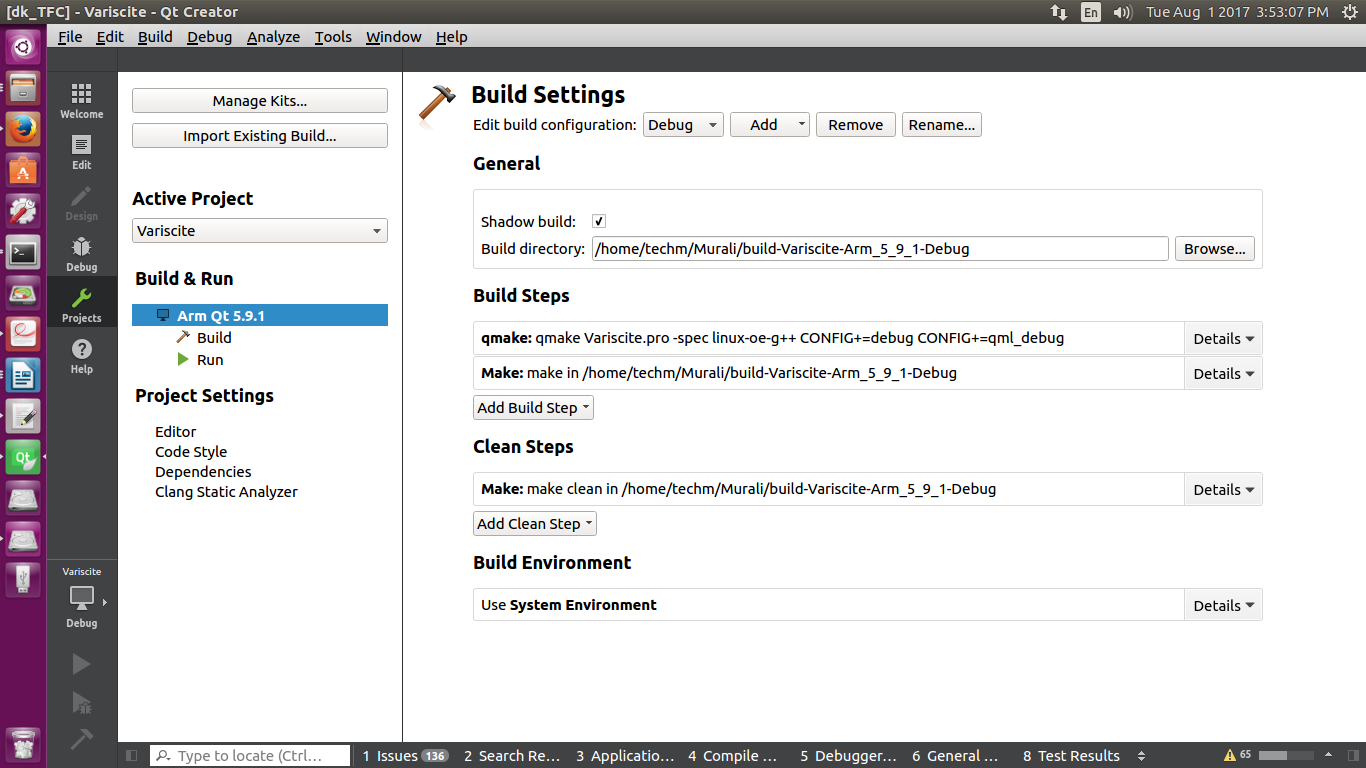
OE\_QMAKE\_RANLIB=arm-fslc-linux-gnueabi-ranlib

OE\_QMAKE\_STRIP=arm-fslc-linux-gnueabi-strip

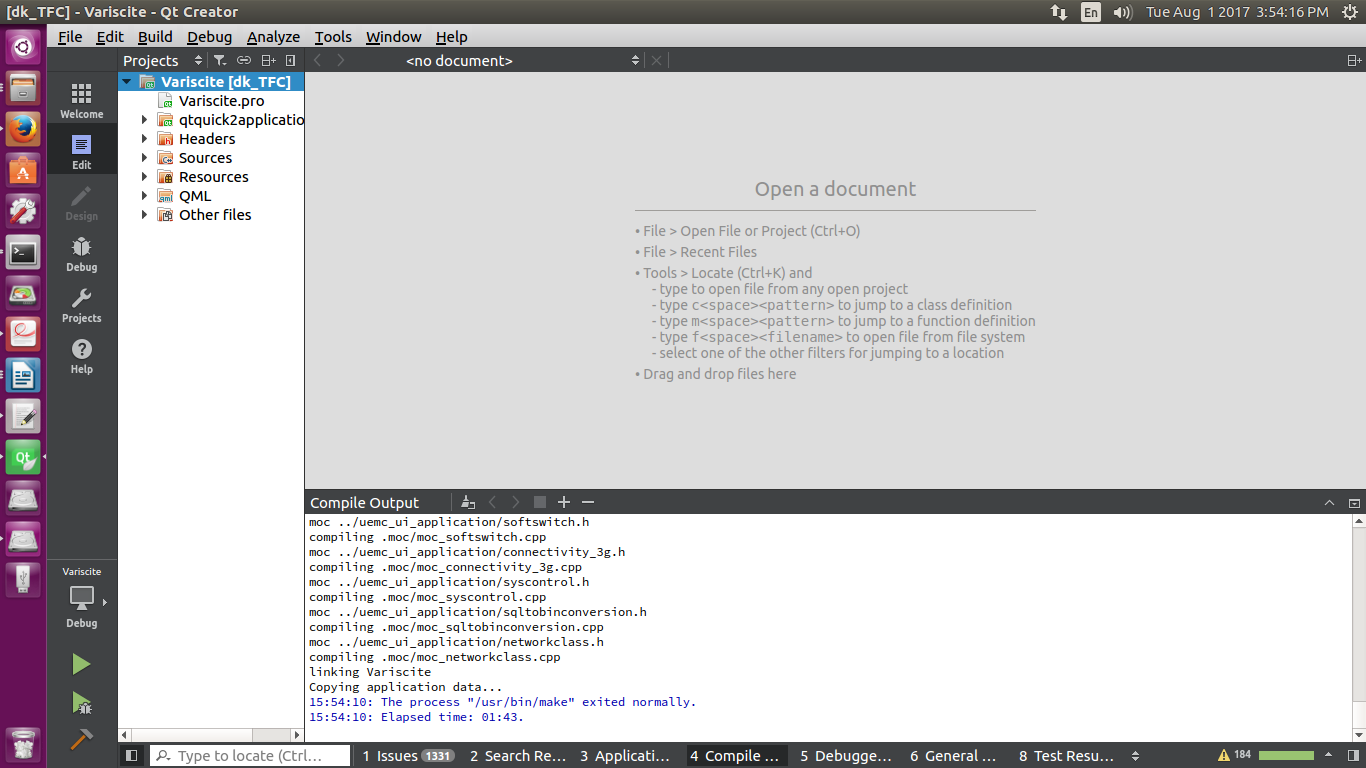
## Building a Project

We should now be able to build and run an application for our Yocto embedded target.

click on Projects=>Open Project and select your project’s .pro file. You'll now be at the Configure Project screen. Select the appropriate kit for your Yocto environment (I called mine "Arm Qt 5.9.1") and click on Configure Project.



You can change some of the project settings if desired, but with the defaults you should now be able to cross-compile the application The Compile Output pane will show the application being cross-compiled.



If it succeeds, start your target. Click on the Run (green arrow) icon and the application should be deployed and run on the target system.

