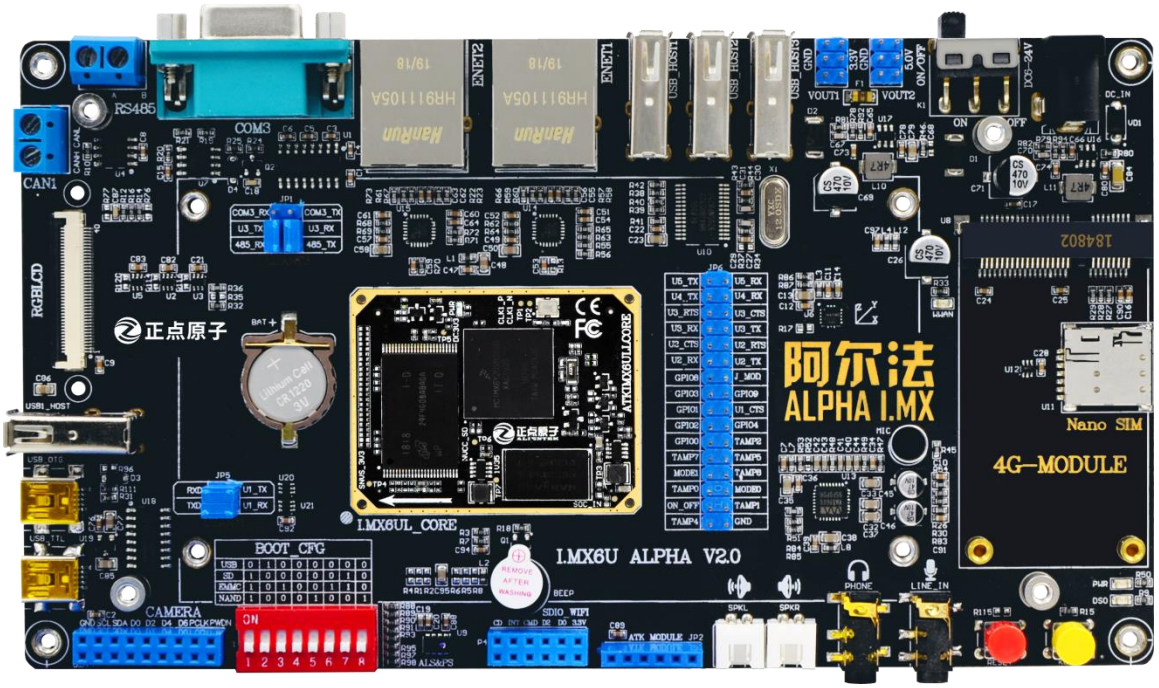
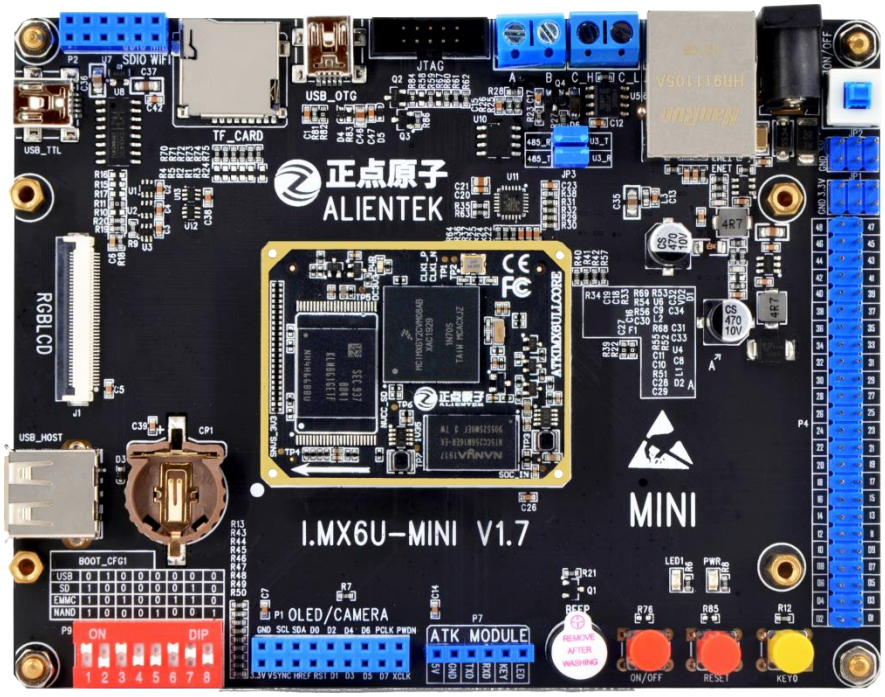
【正点原子】I.MX6U

移植 Qt5.12.9 V1.1





正点原子

正点原子公司名称 ：广州市星翼电子科技有限公司

原子哥在线教学平台：www.yuanzige.com

开源电子网 / 论坛 ：http://www.openedv.com/forum.php

正点原子淘宝店铺 ：https://openedv.taobao.com

正点原子官方网站 ：www.alientek.com

正点原子 B 站视频 ：https://space.bilibili.com/394620890

电话：020-38271790 传真：020-36773971

请关注正点原子公众号，资料发布更新我们会通知。

请下载原子哥 APP，数千讲视频免费学习，更快更流畅。

扫码关注正点原子公众号

扫码下载“原子哥”APP

文档更新说明

版本	版本更新说明	负责人	校审	发布日期
V1.0	初稿：	正点原子 Linux 团队	正点原子 Linux 团队	2020.11.21
V1.1	1. 修改 4.2 小节，添加 libts.so 的环境变量。 2. 修改 3.2 和 3.3 小节，提供脚本下载的链接。 3. 修改文档格式。	正点原子 Linux 团队	正点原子 Linux 团队	2021.06.19

目录

前言

由于正点原子 I.MX6U 使用 Qt5.6.2 版本作为出厂系统 Qt 版本有一段时间了，但是 Qt5.6.2 版本还是过低，Qt 的一些新功能新特性在低版本无法使用。在 2020 年 11 月 20 日以后，我们正点原子出厂系统 Qt 版本已经升级到 Qt5.12.9 LTS（长期支持版本，Qt 官方目前还在更新）。若使用出厂文件系统则不需要再移植 Qt 了!此文档是为了移植 Qt 到其他不含 Qt 的文件系统上。日后正点原子主要使用或提供 Qt4.8.4（老版本 Qt4 客户需要使用）和 Qt5.12.9 作为长期使用版本。此文档编译出来的 Qt5.12.9（非全部功能）支持 Qt Quick/Qt QML 和 Qt Widget，还支持 Qt sqlite!这些都是用户常用的，正点原子以实际用户需要编写文档，若文档有错漏，请联系本文档编写作者 QQ1252699831 指正错误。

本文档所使用的环境：

- Windows 7 64bits，也适用于 Windows 8-10。不建议用 Windows 32 位来开发，Windows 32 位支持的内存大小有限，系统性能有限。
- Ubuntu16.04，Ubuntu 建议使用 16.04 否则安装及编译环境不一样导致出错，请自行解决!
- 要求读者会使用 FileZilla、WinSCP 及 Windows Git 进行 Ubuntu 与 Windows 间互传文件的方法。

第一章 下载安装通用交叉编译器

本章介绍如何下载安装通用的 ARM 交叉编译器，编译 Qt 源码前必须做这项。为什么需要下载通用的交叉编译器来编译 Qt 源码呢？实际上出厂系统已经自带 Qt 库了，不需要编译 Qt 源码了。使用的是 NXP Yocto 项目里的 arm-poky-linux-gnueabi-gcc 编译器，此编译器已经经过 NXP 封装，不适用于编译 Yocto 项目外的 Qt 源码。在这里我们需要注意。如果是使用通用的交叉编译器用来编译 Qt 源码，就不能与出厂系统的 Qt 混用。也就是说，同一个编译器编译出来的 Qt 库，Qt 应用程序，就应该在一起使用。

1.1 下载通用交叉编译器

我们要编译 Qt 源码，需要使用通用的交叉编译器，这里下载 Linaro 出品的交叉编译器，也就是正点原子 I.MX6U 嵌入式 Linux 驱动开发指南第 4.3 小节里推荐的 4.9 版本的编译器，这里重复写下载的方法。如果已经知道怎么安装或者已经安装可跳过第一章。下面是下载地址。https://releases.linaro.org/components/toolchain/binaries/4.9-2017.01/arm-linux-gnueabihf/。

图 1.1 1 选择对应系统版本的交叉编译器下载

请根据个人 Ubuntu 系统的位数，选择对应版本下载即可。也可以在我们正点原子 I.MX6U 开发板光盘 A-基础资料->5、开发工具->1、交叉编译器下找到下载好的交叉编译器。如下图。

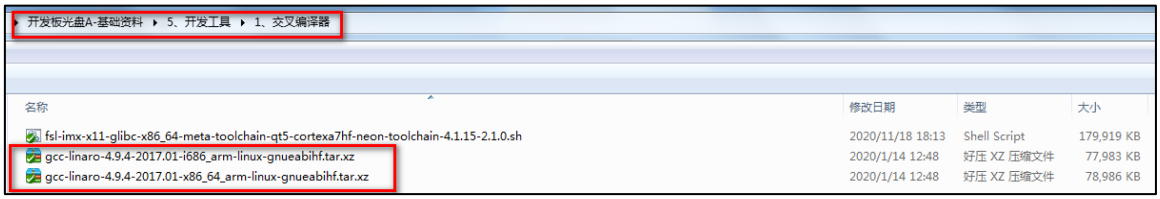


图 1.2 2 资料盘处的交叉编译器

1.2 安装通用交叉编译器

将上面 1.1 小节下载好的通用交叉编译器压缩包拷贝到 Ubuntu 虚拟机，解压进行安装。编者是 64 位的 Ubuntu。所以通用 FileZilla 或者 WinSCP 拷贝 gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz 到 Ubuntu 虚拟机。如下图，编者已经拷贝到“家”目录下。

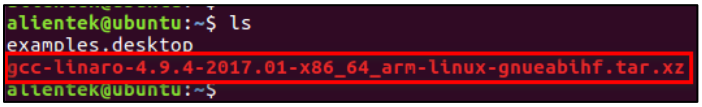


图 1.2 1 拷贝交叉编译器到家目录

在 Ubuntu 目录下创建/usr/local/arm 文件夹，为下面安装到/usr/local/arm 这个文件夹做准备。

```
sudo mkdir /usr/local/arm
```

解压交叉编译器压缩包至/usr/local/arm 目录下，稍等片刻，解压完成如下。

```
sudo tar xf gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz -C /usr/local/arm/
```

```
alientek@ubuntu:~$ sudo mkdir /usr/local/arm
[sudo] alientek 的密码:
alientek@ubuntu:~$ sudo tar xf gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf.tar.xz -C /usr/local/arm/
alientek@ubuntu:~$
```

图 1.2 2 解压交叉编译器

使用 vi 指令编辑/etc/profile 这个文件。

```
sudo vi /etc/profile
```

打开/etc/profile 以后，在末尾添加如下所示内容。

```
export PATH=$PATH:/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin
```

添加完成如下图，保存退出，重启系统。

```
# /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
# and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).

if [ "$PS1" ]; then
  if [ "$BASH" ] && [ "$BASH" != "/bin/sh" ]; then
    # The file bash.bashrc already sets the default PS1.
    # PS1='\h:\w\$ '
    if [ -f /etc/bash.bashrc ]; then
      . /etc/bash.bashrc
    fi
  else
    if [ "`id -u`" -eq 0 ]; then
      PS1='# '
    else
      PS1='$ '
    fi
  fi
fi

if [ -d /etc/profile.d ]; then
  for i in /etc/profile.d/*.sh; do
    if [ -r $i ]; then
      . $i
    fi
  done
  unset i
fi
export PATH=$PATH:/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin
~
```

图 1.2 3 在/etc/profile 下添加全局环境变量

如果不想写在/etc/profile 下，那么每次使用前都要在终端执行一次下面的指令。

```
export PATH=$PATH:/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin
```

1.3 验证通用交叉编译器

要使用此编译器，还要在 Ubuntu 上安装一些库。

```
sudo apt-get install lsb-core lib32stdc++6
```

在 1.2 小节修改环境变量重启系统后，在终端输入 arm-linux-gnueabihf-gcc -v 来查看安装的交叉编译器版本号。看到如下结果，表明成功！

```
arm-linux-gnueabihf-gcc -v
```

```
alientek@ubuntu:~$ arm-linux-gnueabihf-gcc -v
Using built-in specs.
COLLECT_GCC=arm-linux-gnueabihf-gcc
COLLECT_LTO_WRAPPER=/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/../libexec/gcc/arm-linux-gnueabihf/4.9.4/lto-wrapper
Target: arm-linux-gnueabihf
Configured with: /home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/snapshots/gcc-linaro-4.9-2017.01/configure SHELL=/bin/bash --with-mpc=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-linux-gnu --with-mpfr=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-linux-gnu
--with-gmp=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-linux-gnu --with-gnu-as --with-gnu-ld --disable-libmudflap --enable-lto --enable-objc-gc --enable-shared --without-included-gettext --enable-nls --disable-sjlj-exceptions --enable-gnu-unique-object --enable-linker-build-id --disable-libstdcxx-pch --enable-c99 --enable-clocale=gnu --enable-libstdcxx-debug --enable-long-long --with-cloog=no --with-ppl=no --with-isl=no --disable-multilib --with-float=hard --with-mode=thumb --with-tune=cortex-a9 --with-arch=armv7-a --with-fpu=vfpv3-d16 --enable-threads=posix --enable-multiarch --enable-libstdcxx-time=yes --with-build-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/sysroots/arm-linux-gnueabihf --with-sysroot=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-linux-gnu/arm-linux-gnueabihf/libc --enable-checking=release --disable-bootstrap --enable-languages=c,c++,fortran,lto --build=x86_64-unknown-linux-gnu --host=x86_64-unknown-linux-gnu --target=arm-linux-gnueabihf --prefix=/home/tcwg-buildslave/workspace/tcwg-make-release/label/docker-trusty-amd64-tcwg-build/target/arm-linux-gnueabihf/_build/builds/destdir/x86_64-unknown-linux-gnu
Thread model: posix
gcc version 4.9.4 (Linaro GCC 4.9-2017.01)
alientek@ubuntu:~$
```

图 1.3 1 验证交叉编译器

第二章 获取和编译 tslib

要想 Qt 支持触摸还需要编译 tslib，以生成触摸相关插件。Tslib 一般在嵌入式系统时用的多，基本的触摸插件都是它。当然还有 xinput 这样的插件，那是用于桌面系统的。嵌入式 Qt 一般用的都是 Tslib。

2.1 获取 tslib

获取 tslib 的源码，git 地址为 https://github.com/kergoth/tslib。tslib 源码已经放到开发板光盘中路径为：开发板光盘 A-基础资料->1、例程源码->7、第三方库源码->tslib-1.21.tar.bz2。将压缩包发送到 Ubuntu 中并解压，得到名为“tslib-1.21”的目录，此目录下就是 tslib 源码。注意 github 有可能被更新源码导致出现 bug。请使用资料盘里的 tslib-1.21 源码。

拷贝至 Ubuntu 虚拟机目录下，将其解压，得到 tslib-1.21 文件夹。

```
tar xf tslib-1.21.tar.bz2
```

进入 tslib-1.21 文件夹下。

```
cd tslib-1.21
```

```
alientek@ubuntu:~$ cd tslib-1.21/
alientek@ubuntu:~/tslib-1.21$ ls
acinclude.m4  cmake           config.sub    doc          m4           plugins     THANKS
aclocal.m4    CMakeLists.txt  configure     etc          Makefile.am  README      tools
AUTHORS       compile         configure.ac  INSTALL      Makefile.in  README.md   tslib.pc.in
autogen.sh    config.guess    COPYING       install-sh   missing      src
ChangeLog     config.h.in     depcomp       ltmain.sh    NEWS         tests
alientek@ubuntu:~/tslib-1.21$
```

图 2.1 1 进入解压后的 tslib 源码目录下

2.2 编译 tslib

生成 Makefile，还需要安装以下软件。

```
sudo apt-get update
sudo apt-get install autoconf automake libtool
```

执行 autogen.sh 生成 Makefile，以编译源码。

```
alientek@ubuntu:~/tslib-1.21$ ./autogen.sh
libtoolize: putting auxiliary files in '.'.
libtoolize: copying file './ltmain.sh'
libtoolize: putting macros in AC_CONFIG_MACRO_DIRS, 'm4/internal'.
libtoolize: copying file 'm4/internal/libtool.m4'
libtoolize: copying file 'm4/internal/ltoptions.m4'
libtoolize: copying file 'm4/internal/ltsugar.m4'
libtoolize: copying file 'm4/internal/ltversion.m4'
libtoolize: copying file 'm4/internal/lt~obsolete.m4'
configure.ac:58: installing './compile'
configure.ac:7: installing './missing'
plugins/Makefile.am: installing './depcomp'
alientek@ubuntu:~/tslib-1.21$
```

图 2.2 1 生成 Makefile

执行下面的指令指定我们上面第一章安装交叉编译工具与 tslib 编译输出路径/home/alientek/tslib-1.21/arm-tslib。复制时请注意，下面是一条指令不分行，建议分段复制。注意不要多写空格。指令会检查编译器等是否会工作，如下图。

```
./configure --host=arm-linux-gnueabihf ac_cv_func_malloc_0_nonnull=yes --cache-file=arm-linux.cache -prefix=/home/alientek/tslib-1.21/arm-tslib
```

```
alientek@ubuntu:~/tslib-1.21$ ./configure --host=arm-linux-gnueabihf ac_cv_func_malloc_0_nonnull=yes --cache-file=arm-linux.cache -prefix=/home/alientek/tslib-1.2.1/arm-tslib
configure: creating cache arm-linux.cache
checking for a BSD-compatible install... /usr/bin/install -c
checking whether build environment is sane... yes
checking for arm-linux-gnueabihf-strip... arm-linux-gnueabihf-strip
checking for a thread-safe mkdir -p... /bin/mkdir -p
checking for gawk... gawk
checking whether make sets $(MAKE)... yes
checking whether make supports nested variables... yes
checking whether make supports nested variables... (cached) yes
checking build system type... x86_64-pc-linux-gnu
checking host system type... arm-unknown-linux-gnueabihf
checking for arm-linux-gnueabihf-gcc... arm-linux-gnueabihf-gcc
checking whether the C compiler works... yes
checking for C compiler default output file name... a.out
checking for suffix of executables...
checking whether we are cross compiling... yes
checking for suffix of object files... o
checking whether we are using the GNU C compiler... yes
checking whether arm-linux-gnueabihf-gcc accepts -g... yes
checking for arm-linux-gnueabihf-gcc option to accept ISO C89... none needed
checking whether arm-linux-gnueabihf-gcc understands -c and -o together... yes
checking for style of include used by make... GNU
checking dependency style of arm-linux-gnueabihf-gcc... none
checking how to run the C preprocessor... arm-linux-gnueabihf-gcc -E
checking whether the C compiler supports -fvisibility=hidden... yes
checking whether to use -fvisibility=hidden... yes
checking whether ln -s works... yes
checking whether make sets $(MAKE)... (cached) yes
checking how to print strings... printf
checking for a sed that does not truncate output... /bin/sed
checking for grep that handles long lines and -e... /bin/grep
checking for egrep... /bin/grep -E
checking for fgrep... /bin/grep -F
checking for ld used by arm-linux-gnueabihf-gcc... /usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/bin/ld
checking if the linker (/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/bin/ld) is GNU ld... yes
checking for BSD- or MS-compatible name lister (nm)... /usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-nm -B
checking the name lister (/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-nm -B) interface... BSD nm
checking the maximum length of command line arguments... 1572864
checking how to convert x86_64-pc-linux-gnu file names to arm-unknown-linux-gnueabihf format... func_convert_file_noop
checking how to convert x86_64-pc-linux-gnu file names to toolchain format... func_convert_file_noop
checking for /usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/arm-linux-gnueabihf/bin/ld option to reload object files... -r
checking for arm-linux-gnueabihf-objdump... arm-linux-gnueabihf-objdump
checking how to recognize dependent libraries... pass_all
checking for arm-linux-gnueabihf-dlltool... no
checking for dlltool... no
checking how to associate runtime and link libraries... printf %s\n
checking for arm-linux-gnueabihf-ar... arm-linux-gnueabihf-ar
checking for archiver @FILE support... @
checking for arm-linux-gnueabihf-strip... (cached) arm-linux-gnueabihf-strip
checking for arm-linux-gnueabihf-ranlib... arm-linux-gnueabihf-ranlib
checking command to parse /usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-nm -B output from arm-linux-gnueabihf-gcc object... ok
checking for sysroot... no
```

图 2.2 2 配置 tslib

直接执行 make 编译及 make install 安装。

```
make
make install
```

编译出现警告不用理会以，安装完成后在当前目录下生成一个 arm-tslib 目录。也是我们上面指令的编译输出目录路径。可以查看生成目录下有以下内容。

```
alientek@ubuntu:~/tslib-1.21$ ls arm-tslib/
bin  etc  include  lib  share
alientek@ubuntu:~/tslib-1.21$
```

图 2.2 3 编译完成，arm-tslib 目录下的产物

查看编译的 tslib 文件类型，使用 file 指令。编译出来的 tslib 应为 ARM 格式，不能为 X86 格式，如果是 X86 格式就不能用于 ARM 开发板上了。那么说明您上一章的 ARM 交叉编译器环境没生效。请删除重新编译！

```
file bin/ts_calibrate
```

```
alientek@ubuntu:~/tslib-1.21/arm-tslib$ file bin/ts_calibrate
bin/ts_calibrate: ELF 32-bit LSB executable, ARM, EABI5 version 1 (SYSV), dynamically linked, interpreter
/lib/ld-linux-armhf.so.3, for GNU/Linux 2.6.31, BuildID[sha1]=e361dcb49b0059b35788de397d1a483f772f1c57, no
t stripped
alientek@ubuntu:~/tslib-1.21/arm-tslib$
```

ARM类型，不能够为X86_64类型

图 2.2 4 查看 tslib 产物的格式类型

第三章 编译 ARM 平台 Qt5.12.9 源码

本章简介如何下载 Qt 源码。编译出相应的 Qt 库。在 Windows 平台下的库，后缀一般都是 dll，Linux 平台的库后缀都是 so。我们能看到的这些库都叫动态库。运行 Qt 应用程序就需要链接到这些动态库上面。正是因为有这些库的存在，Qt 应用程序才能正常运行。

3.1 下载 Qt5.12.9 源码

我们可以在 Qt 下载地址 https://download.qt.io/找到 https://download.qt.io/archive/qt/5.12/5.12.9/single/，进入下载页面如下。

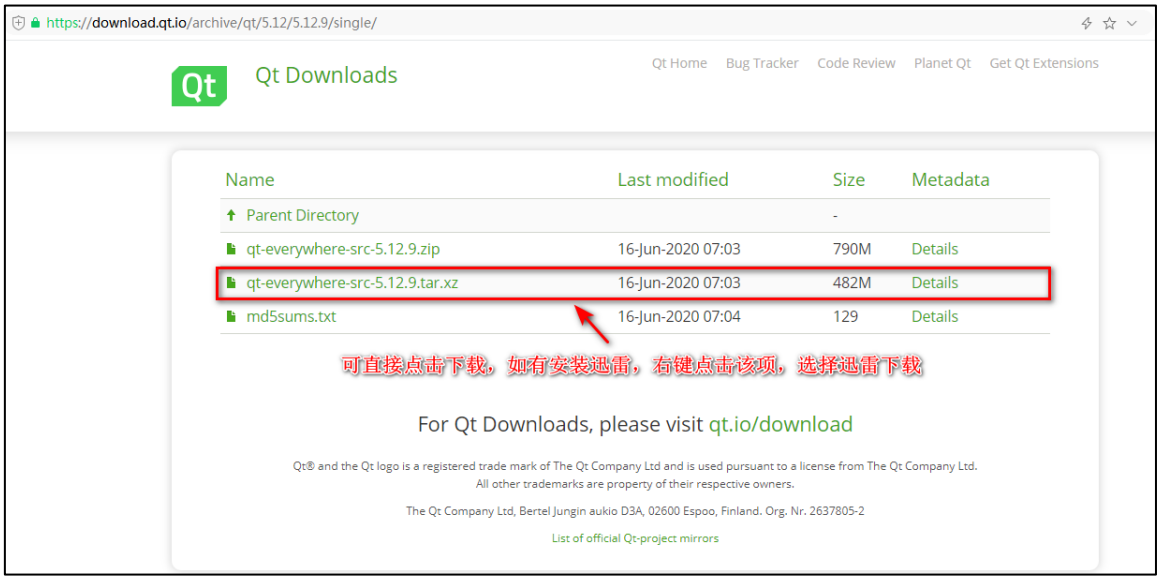


图 3.1 1 下载 Qt5.12.9 源码

在 Windows 上下载好上面 qt-everywhere-src-5.12.9.tar.xz 源码拷贝到 Ubuntu 虚拟机下，或者直接右键上面的下载项，复制下载地址链接，在 Ubuntu 下直接使用 wget 指令下载。

在 Ubuntu 使用 wget 指令下载 qt-everywhere-src-5.12.9.tar.xz 源码如下。如果觉得慢，就在 Windows 下复制下载链接地址使用迅雷下载再拷贝到 Ubuntu 虚拟机吧！

```
wget https://download.qt.io/archive/qt/5.12/5.12.9/single/qt-everywhere-src-5.12.9.tar.xz
```

图 3.1 2 使用 wget 指令下载 Qt5.12.9 源码

解压下载好的 qt-everywhere-src-5.12.9.tar.xz 源码压缩包。解压时间较长，请耐心等待。

```
tar xf qt-everywhere-src-5.12.9.tar.xz
```

在当前目录出现 qt-everywhere-src-5.12.9 目录夹，此文件夹就是我们解压后的目录。进入此目录。

```
cd qt-everywhere-src-5.12.9/
ls
```

图 3.1 3 解压后的 Qt5.12.9 源码包

3.2 修改 qmake.conf

编辑 mkspecs/qws/linux-arm-gnueabi-g++/qmake.conf，配置相关编译器及路径。由于用户经常复制文档错误，指的是格式或者手敲错误。所以编者将这些配置上传到 gitee 上，提供给用户

下载。路径如下。https://gitee.com/QQ1252699831/qt5.12.9-conf.git。可以直接在 gitee 上下载这个 qmake.conf。

```
vi qtbase/mkspecs/linux-arm-gnueabi-g++/qmake.conf
```

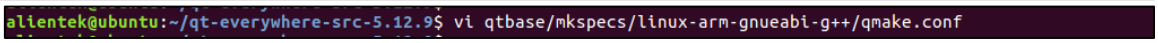


图 3.2 1 配置 qmake.conf

修改成如下，红色加粗部分就是要修改的地方。

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

MAKEFILE_GENERATOR      = UNIX
CONFIG                  += incremental
QMAKE_INCREMENTAL_STYLE = sublib

QT_QPA_DEFAULT_PLATFORM = linuxfb
QMAKE_CFLAGS += -O2 -march=armv7-a -mtune=cortex-a7 -mfpu=neon -mfloat-abi=hard
QMAKE_CXXFLAGS += -O2 -march=armv7-a -mtune=cortex-a7 -mfpu=neon -mfloat-abi=hard

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)
```

3.3 配置编译选项

查看编译选项，输入./configure -help 指令，查看可配置选项。

```
./configure -help
```

由于配置较长，按需要编译，编者总结了配置项，写了一个脚本。由于用户经常复制文档错误，指的是 PDF 格式或者手敲错误。所以编者将这些配置上传到 gitee 上，提供给用户下载。路径如下。https://gitee.com/QQ1252699831/qt5.12.9-conf.git。可以直接在 gitee 上下载这个 autoconfigure.sh。下载之后修改里面的路径，改为自己的即可！

```
vi autoconfigure.sh
```

在这个 autoconfigure.sh 添加以下内容。复制时注意，每行前面不要留空格。

```
./configure -prefix /home/alientek/qt-everywhere-src-5.12.9/arm-qt \
-opensource \
-confirm-license \
-release \
-strip \
-shared \
-xplatform linux-arm-gnueabi-g++ \
-optimized-qmake \
-c++std c++11 \
--rpath=no \
-pch \
-skip qt3d \
-skip qtactiveqt \
-skip qtandroidextras \
-skip qtcanvas3d \
-skip qtconnectivity \
-skip qtdatavis3d \
-skip qtdoc \
-skip qtgamepad \
-skip qtlocation \
-skip qtmacextras \
-skip qtnetworkauth \
-skip qtpurchasing \
-skip qtremoteobjects \
-skip qtscript \
-skip qtscxml \
-skip qtsensors \
-skip qtspeech \
-skip qtsvg \
-skip qttools \
-skip qttranslations \
-skip qtwayland \
-skip qtwebengine \
-skip qtwebview \
-skip qtwinextras \
-skip qtx11extras \
-skip qtxmlpatterns \
-make libs \
-make examples \
-nomake tools -nomake tests \
-gui \
-widgets \
```

```
-dbus-runtime \
--glib=no \
--iconv=no \
--pcre=qt \
--zlib=qt \
-no-openssl \
--freetype=qt \
--harfbuzz=qt \
-no-opengl \
-linuxfb \
--xcb=no \
-tslib \
--libpng=qt \
--libjpeg=qt \
--sqlite=qt \
-plugin-sql-sqlite \
-I/home/alientek/tslib-1.21/arm-tslib/include \
-L/home/alientek/tslib-1.21/arm-tslib/lib \
-recheck-all
```

其中红色加粗部分为编者个人的路径，请修改成个人路径。

（1） /home/alientek/ qt-everywhere-src-5.12.9/arm-qt 为编译输出路径。

（2） /home/alientek/tslib-1.21/arm-tslib/include 为第二章 tslib 头文件路径。

（3） /home/alientek/tslib-1.21/arm-tslib/lib 为第二章 tslib 相关库文件路径。

赋予脚本可执行权限。

```
chmod +x autoconfigure.sh
```

直接执行 autoconfigure.sh。配置需要一段时间，请耐心等待！

```
sudo apt-get install g++ // 配置前请先安装 g++
./autoconfigure.sh
```

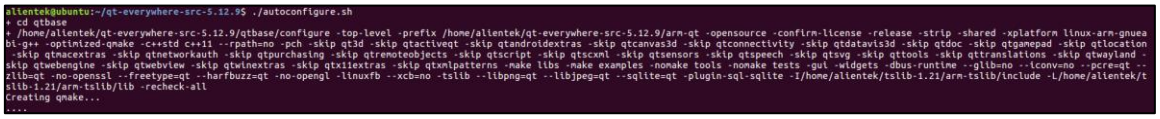


图 3.3 1 正在配置编译项

配置完成如下。若未见此信息，请检查个人配置。切记是使用 Ubuntu16，如果是 Ubuntu18 以上，可能出现 C++11 Error，这是我们正点原子忠实用户粉丝反馈的，粉丝解决方法是出现这种错误后，重新解压一次 Qt 源码，重新配置就不会出现 c++11 Error，特此说明。

图 3.3 2 配置成功

3.4 开始编译 Qt5.12.9 源码

直接执行 make 开始编译。

time (make -j 16) // -j 16 代表最多允许 16 条编译指令同时运行，参数 16，一般为个人分配给虚拟机的核心数的 2 倍，前面加个 time 是测试编译时间。

编译完成如下图，编译时长由个人计算机速度及分配给虚拟机核心数决定（编者编译了 14 分钟）。如按上面操作，遇到编译 Qt 源码出现错误，请重新配置！有可能是个人计算机分配给虚拟机内存太低所导致的，既然 Qt 源码能发布，除了配置错误，几乎不可能有编译错误的！经过编者经验和用户反馈，用户编译的时候会出现类似 sub-xx-make-first-ordered 错误。这个原因可能是虚拟机分配的配置太低及用户目录权限的问题。您可以看见编者都是用普通用户在编译的，尽量不要使用 root 用户和 sudo 权限来编译！除非您比较懂。

编译完成如下图。

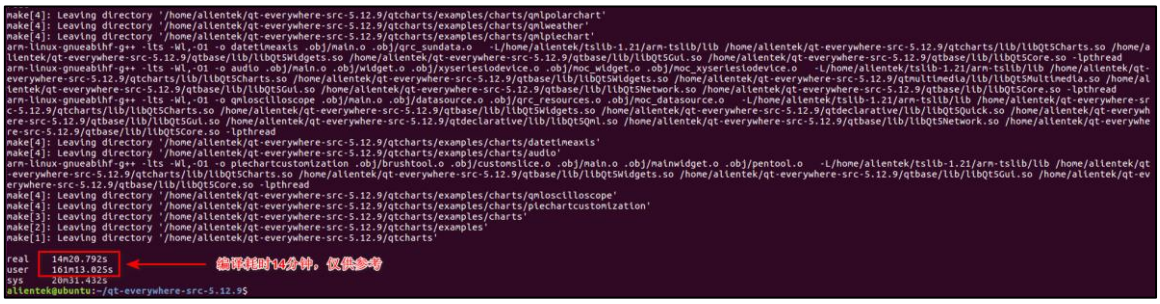


图 3.4 1 Qt5.12.9 源码编译完成

执行安装指令后，查看安装后的内容。

```
make install
ls arm-qt
```

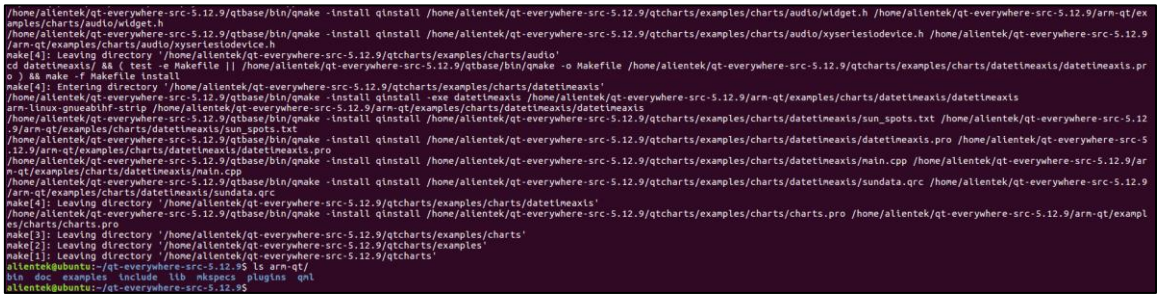

/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/audio/widget.h /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/audio/widget.h
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/audio/xyseriesiodevice.h /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/audio/xyseriesiodevice.h
make[4]: Leaving directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/audio'
cd datetimeaxis/ && (test -e Makefile || /home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -o Makefile /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis/datetimeaxis.pro) && make -f Makefile install
make[4]: Entering directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis'
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall -exe datetimeaxis /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/datetimeaxis
arm-linux-gnueabihf-strip /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/datetimeaxis
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis/sun_spots.txt /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/sun_spots.txt
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis/datetimeaxis.pro /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/datetimeaxis.pro
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis/main.cpp /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/main.cpp
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis/sundata.qrc /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/datetimeaxis/sundata.qrc
make[4]: Leaving directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/datetimeaxis'
/home/alientek/qt-everywhere-src-5.12.9/qtbase/bin/qmake -install qinstall /home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts/charts.pro /home/alientek/qt-everywhere-src-5.12.9/arm-qt/examples/charts/charts.pro
make[3]: Leaving directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples/charts'
make[2]: Leaving directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts/examples'
make[1]: Leaving directory '/home/alientek/qt-everywhere-src-5.12.9/qtcharts'
alientek@ubuntu:~/qt-everywhere-src-5.12.9$ ls arm-qt/
bin doc examples include lib mkspecs plugins qml
alientek@ubuntu:~/qt-everywhere-src-5.12.9$

图 3.4 2 安装完成

第四章 移植 Qt 到文件系统

上面我们终于编译好了 tslib 和 Qt 源码。我们需要移植这些库都开发板系统上才能运行 Qt 应用程序！这里特别说明一下。因为正点原子的出厂系统自带 Qt 库和 tslib，是不需要额外移植了。不能将自己编译的 Qt 库移植到出厂系统下！因为如果您不了解出厂系统的环境，就会导致移植的 Qt 库与出厂的 Qt 冲突！导致报错！所以我们需要移植到一个没有 Qt 库的文件系统里。正点原子 Linux 驱动指南里有编译好的 busybox 文件系统。刚好这个文件系统没有 Qt 库，所以我们直接移植到这个 busybox 文件系统里做实验！

4.1 烧写文件系统

这里使用【正点原子】I.MX6U 嵌入式 Linux 驱动开发指南 V1.x.pdf 所构建好的 busybox 文件系统（不含 Qt 的文件系统），文件系统路径为开发板光盘 A-基础资料->8、系统镜像->2、教程系统镜像->3、文件系统->2、busybox-1.29.0 根文件系统下的 rootfs.tar.bz2。将此文件系统替换到开发板光盘 A-基础资料->5、开发工具->4、正点原子 MFG_TOOL 出厂固件烧录工具->mfgtool->Profiles->Linux->OS Firmware->files->filesystem 下的 rootfs.tar.bz2 (替换前请备份好原来的文件系统！)。特别注意，经过用户反馈，有些用户为了图省事，通过 NFS 挂载文件系统的方式来运行 Qt 程序。此方法运行 Qt 程序可能导致网络中断!因为 Qt 运行会耗一定的性能，而挂载文件系统也会消耗掉一定的性能。所以可能会出现网络中断，串口终端没反应，Qt 程序运行缓慢等情况。所以建议读者烧写系统到板子在的存储或者 TF 卡上。这样更能真实反映 Qt 运行的性能!

然后烧写文件系统到 eMMC 核心板或者 NandFlash 核心板。烧写方法请看【正点原子】I.MX6U 用户快速体验 V1.x.pdf 第 2.2 小节固化系统。

4.2 移植 tslib 到文件系统

使用下面的指令打包 2.2 小节编译安装好的 arm-tslib 文件夹打包成 tar.bz2 格式，打包以防止文件丢失。

```
tar -jcf ./arm-tslib.tar.bz2 arm-tslib
```

```
alientek@ubuntu:~/tslib-1.21$ tar -jcf ./arm-tslib.tar.bz2 arm-tslib
alientek@ubuntu:~/tslib-1.21$ ls
acinclude.m4        autom4te.cache  config.h.in     COPYING     ltmain.sh    plugins     tools
aclocal.m4          ChangeLog       config.h.in~    depcomp     m4           README      tslib.pc
arm-linux.cache     cmake           config.log      doc         Makefile     README.md   tslib.pc.in
arm-tslib           CMakeLists.txt  config.status   etc         Makefile.am  src
arm-tslib.tar.bz2   compile         config.sub      INSTALL     Makefile.in  stamp-h1
AUTHORS             config.guess    configure       install-sh  missing      tests
autogen.sh          config.h        configure.ac    libtool     NEWS         THANKS
alientek@ubuntu:~/tslib-1.21$ _
```

图 4.2 1 打包 arm-tslib

然后使用 U 盘拷贝 arm-tslib.tar.bz2，在 USB 接口插上 U 盘，手动挂载后，在开发板根文件系统上解压到/usr/lib/目录下。如下图，编者已经解压到/usr/lib 目录下。

手动挂载 U 盘指令参考。

```
mkdir mnt // 创建一个目录，让 U 盘挂载到此目录下，我们就可以访问这个目录了!注意，U 盘的格式需要为 FAT32 格式，或者常见的 Linux 类型格式，不能为 NTFS 格式。
```

挂载 U 盘的目录，sda1 为 U 盘的分区，不要写成 sda!sda 只是设备名。

```
mount /dev/sda1 /mnt
```

进入 U 盘的挂载目录后，将 arm-tslib.tar.bz2 解压到/usr/lib 目录下。

```
tar xf arm-tslib.tar.bz2 -C /usr/lib
```

```
/usr/lib # ls
arm-tslib                libnl-genl-3.la
engines-1.1              libnl-genl-3.so
libBrokenLocale.a        libnl-genl-3.so.200
libBrokenLocale.so       libnl-genl-3.so.200.18.0
libBrokenLocale_p.a      libnl-idiag-3.a
libBrokenLocale_pic.a    libnl-idiag-3.la
libanl.a                 libnl-idiag-3.so
libanl.so                libnl-idiag-3.so.200
libanl_p.a               libnl-idiag-3.so.200.18.0
libanl_pic.a             libnl-nf-3.a
```

图 4.2 2 解压打包的 arm-tslib 到/usr/lib 目录下

➢ 配置 tslib 的环境变量

由于上面烧写的文件系统已经移植过 tslib，已经在/etc/profile 文件配置过 tslib 的环境变量。所以我们的要重新配置我们自己的 tslib 环境变量，指定我们编译的 tslib 所在文件夹。避免和文件系统已经存在的 tslib 冲突。（那究竟能不能直接用文件系统已经存在的 tslib 呢？这个自行测试，怕用户在第二章移植 tslib 用了不同版本，所以我们还是自己指定我们移植的 tslib 吧）。编辑/etc/profile 文件，修改成以下内容。如下图红色框部分。“export TSLIB_CALIBFILE=/etc/pointercal”这项是电阻屏专用，如果是电容屏，可不用加这项。电容屏加了这项，如果使用 ts_calibrate 校准后会生成/etc/pointercal 文件，请把它删除！否则可以触摸不准确，因为电容屏不需要校准。注意，下面指令已经添加了 export LD_PRELOAD=$TSLIB_ROOT/lib/libts.so。经过有些用户反馈，用了自己制作的文件系统，不知道为何就找不到 libts.so。所以我们需要添加上 export LD_PRELOAD=$TSLIB_ROOT/lib/libts.so 这个环境变量！

```
export TSLIB_ROOT=/usr/lib/arm-tslib
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CONFFILE=$TSLIB_ROOT/etc/ts.conf
export TSLIB_PLUGINDIR=$TSLIB_ROOT/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal
export LD_PRELOAD=$TSLIB_ROOT/lib/libts.so
```

```
#!/bin/sh
LD_LIBRARY_PATH=/lib:/usr/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

export TERM=vt100
export TERMINFO=/usr/share/terminfo

export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_ROOT=/usr/lib/arm-tslib
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CONFFILE=$TSLIB_ROOT/etc/ts.conf
export TSLIB_PLUGINDIR=$TSLIB_ROOT/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal

export ALSA_CONFIG_PATH=/usr/share/arm-alsa/alsa.conf
```

图 4.2 31 配置 tslib 全局环境变量

➢ 测试 tslib

使能 tslib 的配置在/etc/profile 的环境变量，下次开机不用使能，开机会自动使能这个环境变量。

```
source /etc/profile                    // 使能环境变量
/usr/lib/arm-tslib/bin/ts_test         // 运行 ts_test 测试触摸是否正常，点击界面的 Draw 测试
```

触摸屏上点击 Draw 测试能够画出线条，且位置准确，说明 tslib 配置正常。按 Ctrl + c 结束 ts_test 指令。下图为 7 寸 800x480 分辨率的电容屏，使用 tslib 测试效果。

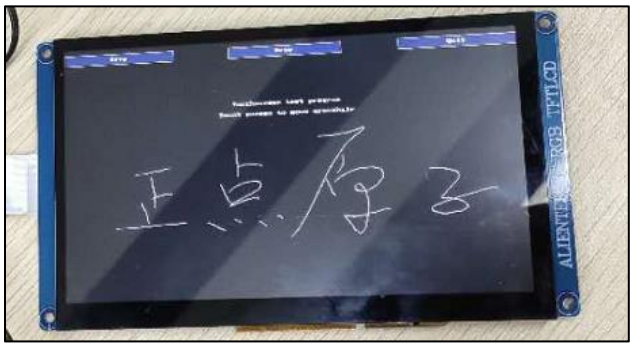


图 4.2 4 7 寸屏 800x480 的触摸效果

4.3 移植 Qt 到文件系统

使用下面的指令打包 3.4 小节编译安装好的 arm-qt 文件夹打包成 tar.bz2 格式，打包以防止文件丢失。

```
tar -jcf ./arm-qt.tar.bz2 arm-qt
```

然后使用 U 盘拷贝 arm-qt.tar.bz2，在 USB 接口插上 U 盘（参考上一小节的方法挂载 U 盘），手动挂载后，在开发板根文件系统上解压到/usr/lib/目录下。如下图，编者已经解压到/usr/lib 目录下。

```
tar xf arm-qt.tar.bz2 -C /usr/lib
```

```
/usr/lib # ls
arm-qt                  libnl-genl-3.la
arm-tslib               libnl-genl-3.so
engines-1.1             libnl-genl-3.so.200
libBrokenLocale.a       libnl-genl-3.so.200.18.0
libBrokenLocale.so      libnl-idiag-3.a
```

图 4.3 1 解压打包 arm-qt 到/usr/lib 下

➢ 配置 Qt5 的环境变量

编辑/etc/profile，在末尾添加以下内容。如下图红色框内。注意要改为个人实际的路径。要想 Qt 程序显示中文，请自行将 windows 下的（路径 C:\Windows\Fonts）下的中文字库放到新建一个/usr/share/fonts/目录下就可以了。若例程有使用到字符，会显示找到不字库。注意 Windwos 的字库仅为个人学习使用，不要用于商业用途！有版权的！

```
export QT_ROOT=/usr/lib/arm-qt
export QT_QPA_GENERIC_PLUGINS=tslib:/dev/input/event1
export QT_QPA_FONTDIR=/usr/share/fonts
export QT_QPA_PLATFORM_PLUGIN_PATH=$QT_ROOT/plugins
export QT_QPA_PLATFORM=linuxfb:tty=/dev/fb0
export QT_PLUGIN_PATH=$QT_ROOT/plugins
export LD_LIBRARY_PATH=$QT_ROOT/lib:$QT_ROOT/plugins/platforms
export QML2_IMPORT_PATH=$QT_ROOT/qml
export QT_QPA_FB_TSLIB=1
```

```
#!/bin/sh
LD_LIBRARY_PATH=/lib:/usr/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

export TERM=vt100
export TERMINFO=/usr/share/terminfo

export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_ROOT=/usr/lib/arm-tslib
export TSLIB_CONSOLEDEVICE=none
export TSLIB_FBDEVICE=/dev/fb0
export TSLIB_TSDEVICE=/dev/input/event1
export TSLIB_CONFFILE=$TSLIB_ROOT/etc/ts.conf
export TSLIB_PLUGINDIR=$TSLIB_ROOT/lib/ts
export TSLIB_CALIBFILE=/etc/pointercal

export QT_ROOT=/usr/lib/arm-qt
export QT_QPA_GENERIC_PLUGINS=tslib:/dev/input/event1
export QT_QPA_FONTDIR=/arm-qt/lib/fonts
export QT_QPA_PLATFORM_PLUGIN_PATH=$QT_ROOT/plugins
export QT_QPA_PLATFORM=linuxfb:tty=/dev/fb0
export QT_PLUGIN_PATH=$QT_ROOT/plugins
export LD_LIBRARY_PATH=$QT_ROOT/lib:$QT_ROOT/plugins/platforms
export QML2_IMPORT_PATH=$QT_ROOT/qml
export QT_QPA_FB_TSLIB=1

export ALSA_CONFIG_PATH=/usr/share/arm-alsa/alsa.conf
~
```

图 4.3 2 配置 Qt5 的全局环境变量

➢ 测试 Qt 运行

使能 Qt 的配置在/etc/profile 的环境变量，下次开机不用使能，开机会自动使能这个环境变量。

```
source /etc/profile
/usr/lib/arm-qt/examples/widgets/animation/animatedtiles/animatedtiles  //运行编译的示例
```

7 寸屏 800x480 显示效果如下(拍照效果略差，实际效果很好)，同时点击图中的项，确认触摸正常。Qt 官方例子也十分流畅!开启程序需要提前插鼠标！移植的 Qt 不支持热插拔鼠标(出厂系统支持热插拔)，Qt 官方说明需要 libudev 情况下 Qt 才能支持热插拔，请自行解决。

图 4.3 3 7 寸屏 800x480 Qt5 例程运行效果

第五章 搭建 ARM 平台的 Qt Creator 环境

万里长征，终于到搭建 ARM 平台的 Qt Creator 环境这一步了！我们上面只是移植好了 Qt 到开发板上。我们还需要编写 Qt 应用程序啊，最后交叉编译 Qt 应用程序才能算完整！本章将介绍 Qt Creator 的安装和交叉编译 ARM 平台的 Qt 环境搭建。

5.1 安装 Qt Creator

我们想要在 Ubuntu 平台开发 Qt，那么必须要有相应的 IDE 环境，那么 Qt Creator 就是 Qt 的 IDE(集成环境)，我们就可以使用 Qt Creator 快速编程了。进入网址 http://download.qt.io/archive/qt/5.12/5.12.9/下载 Qt 5.12.9。Qt 官方已经把 Qt Creator 封装在下面这个 qt-opensource-linux-x64-5.12.9.run。我们需要下载这个 qt-opensource-linux-x64-5.12.9.run 直接在 Ubuntu 安装即可。

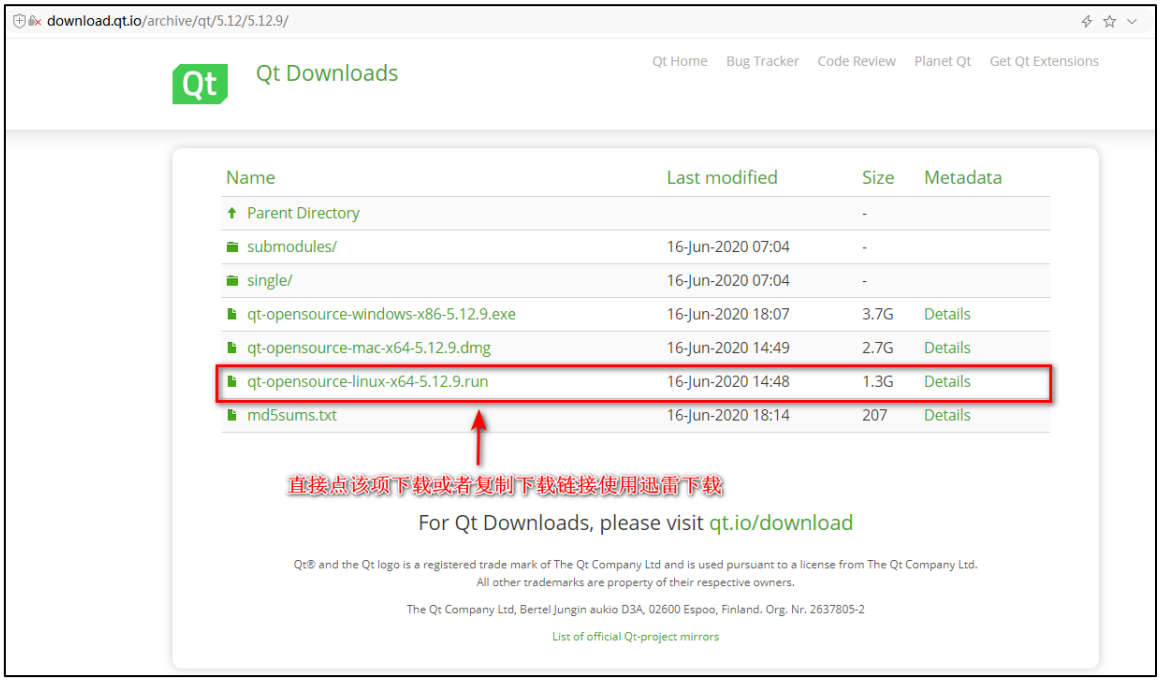


图 5.1 1 点击下载 Qt5.12.9

编者喜欢用指令下载，编者复制上面的下载地址，直接在 Ubuntu 终端使用 wget 指令下载。如下。

```
wget http://download.qt.io/archive/qt/5.12/5.12.9/qt-opensource-linux-x64-5.12.9.run
```

图 5.1 2 使用 wget 指令在 Ubuntu 终端直接下载

下载完成得到 qt-opensource-linux-x64-5.12.9.run 这个文件。

赋予该文件可执行权限，再执行安装。

```
chmod u+x qt-opensource-linux-x64-5.12.9.run
sudo ./qt-opensource-linux-x64-5.12.9.run // 建议加 sudo，否则安装在当前目录下。
```

如无意外，则弹出 Qt 5.12.9 欢迎安装界面。

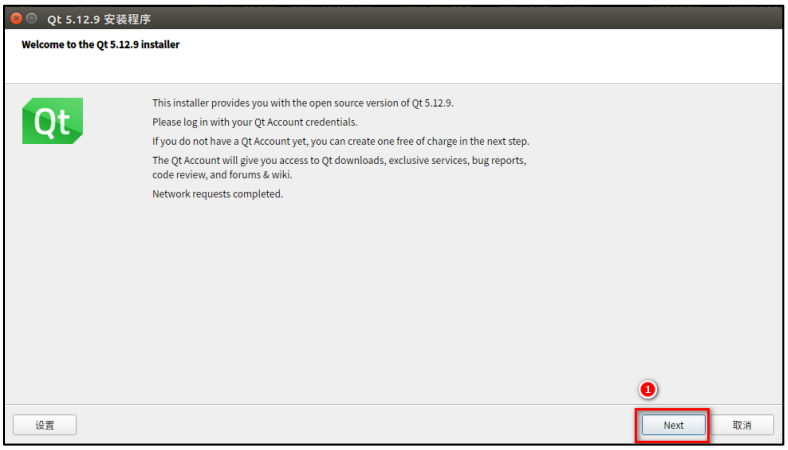


图 5.1 3 欢迎安装界面

这里需要填写 Qt 帐号，去 Qt 官网 https://www.qt.io/注册一个账号。填写后再点击 Next。

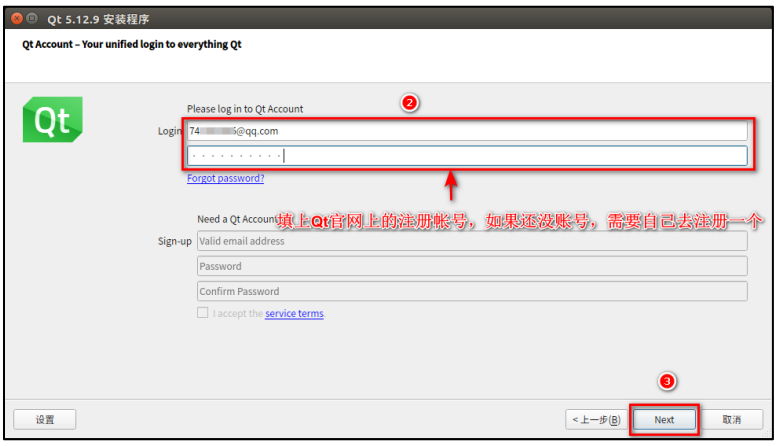


图 5.1 4 填写 Qt 帐号

同意使用条款。按如下步骤操作。

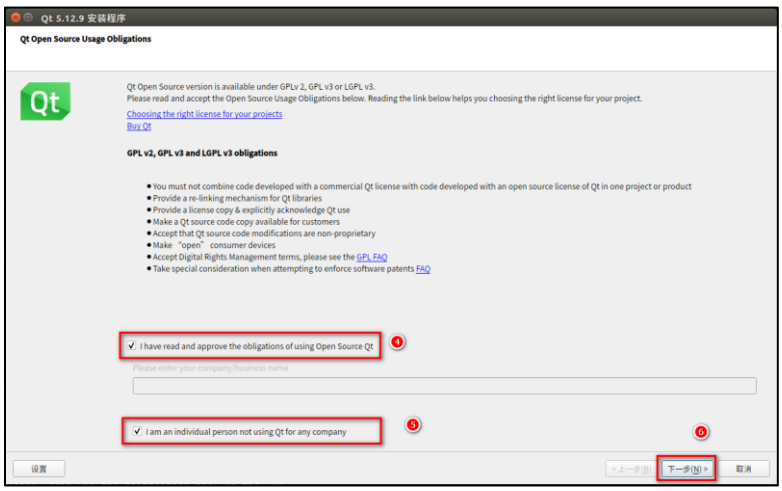


图 5.1 5 同意使用条款

欢迎安装界面，直接点击下一步。

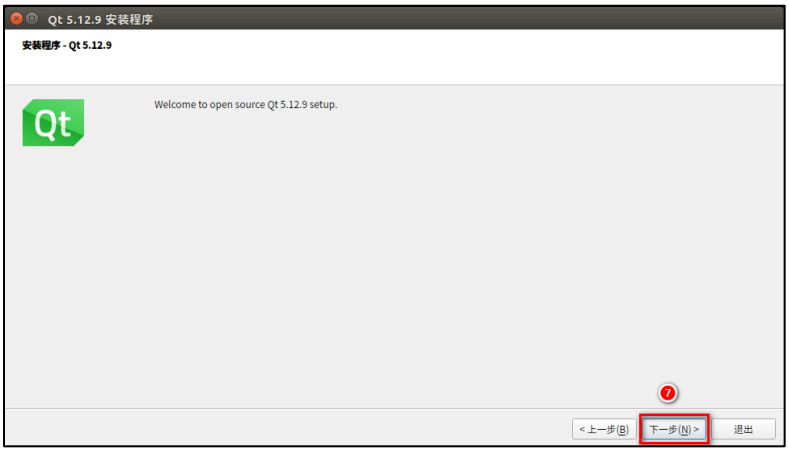


图 5.1 6 欢迎安装界面

选择安装目录，建议默认即可。会安装在/opt 目录下。点击下一步。

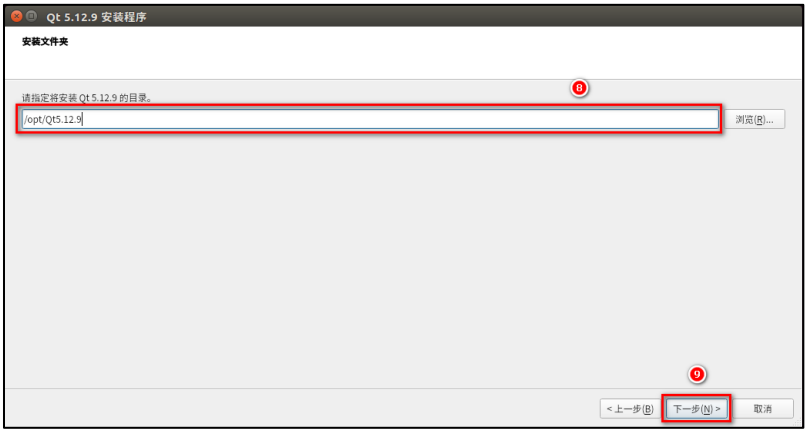


图 5.1 7 默认安装在/opt 目录下

按需要安装，除了安卓选项我们都选择安装。点击下一步。

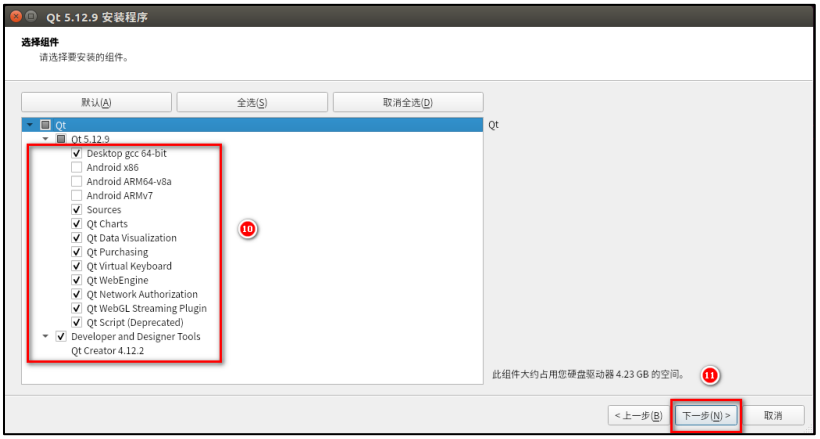


图 5.1 8 勾选安装选项

同意许可协议，点击下一步。

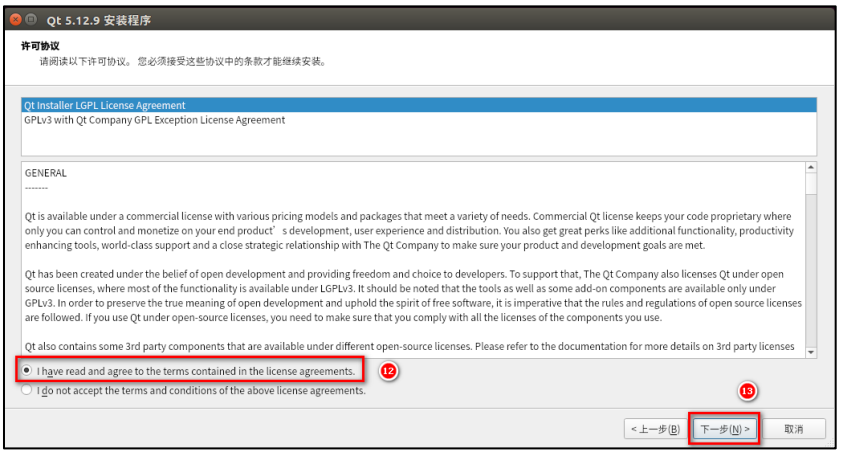


图 5.1 9 同意许可协议

准备安装，点击安装

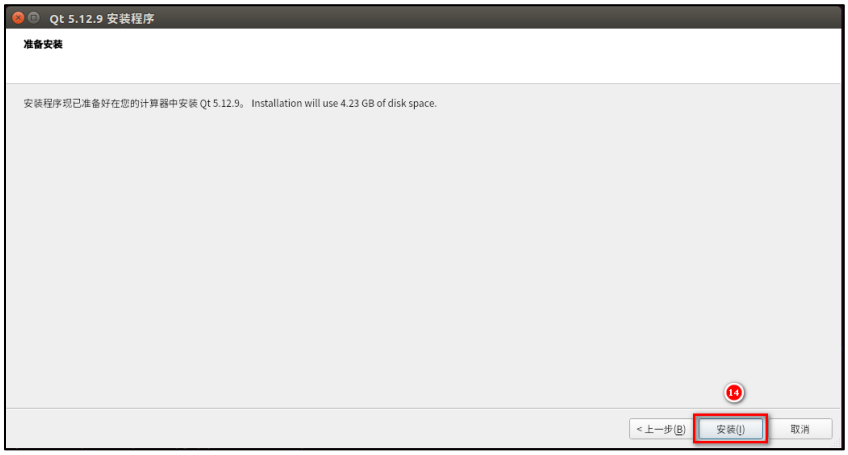


图 5.1 10 开始安装

安装完成，点击完成

图 5.1 11 安装完成

5.2 配置 ARM 平台的 Qt Creator Kits

若安装时和编者安装的目录一样则可以使用如下指令打开 Qt Creator。“&”的作用是后台运行。也可以在左上角的软件中心栏输入搜索出“Qt Creator”图标后再单击打开。

```
/opt/Qt5.12.9/Tools/QtCreator/bin/qtcreator.sh &
```

```
alientek@ubuntu:~$ /opt/Qt5.12.9/Tools/QtCreator/bin/qtcreator.sh &
[1] 10672
alientek@ubuntu:~$
```

图 5.2 1 后台运行 Qt Creator

打开 Qt Creator 界面，找到 Tools（工具）》Options（选项）。

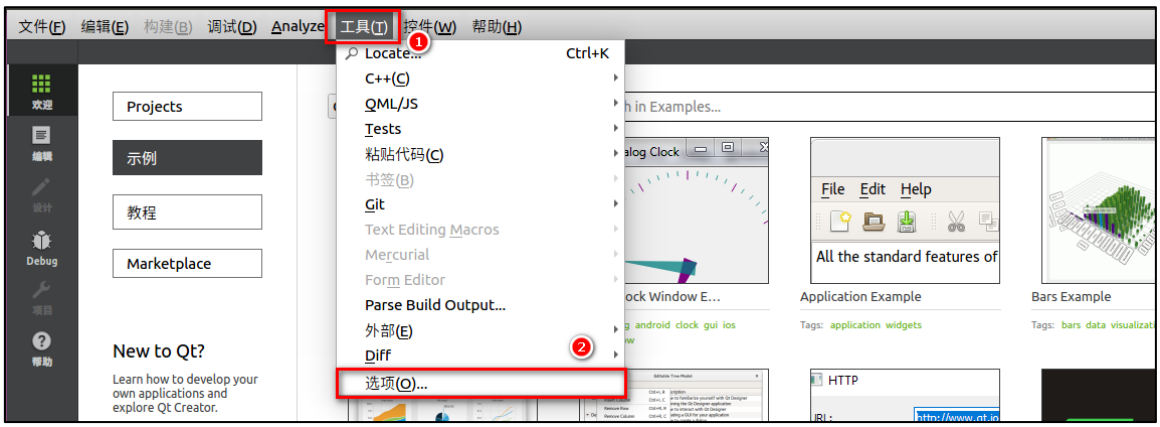


图 5.2 2 打开选项栏

➢ 配置 qmake

按下图步骤，找到我们第 3.4 小节编译出来的 ARM 平台的 Qt 库安装文件夹 arm-qt 下的 qmake。

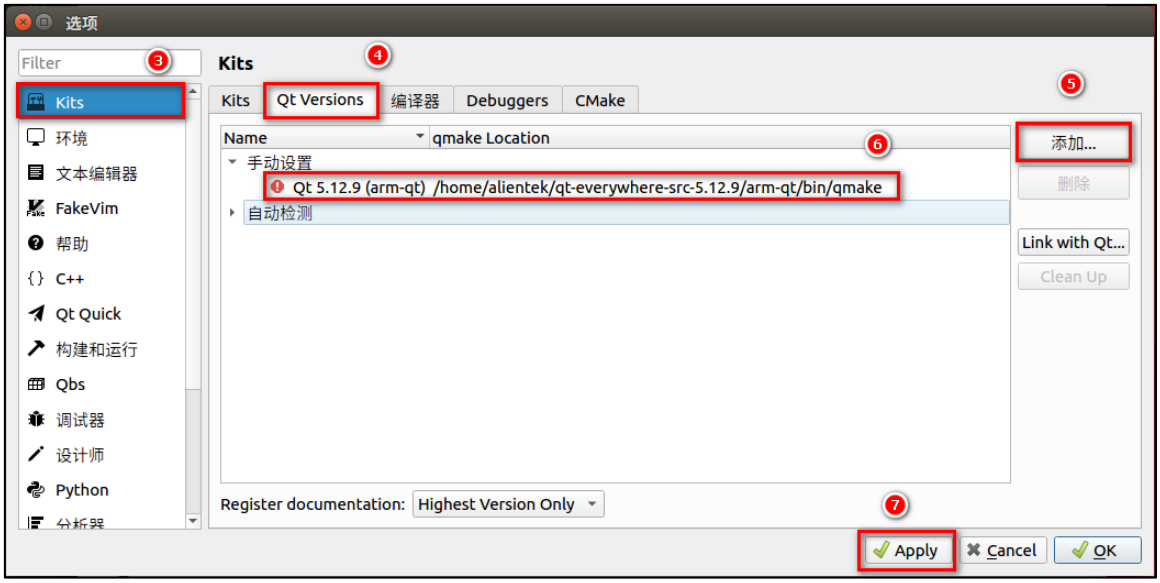


图 5.2 3 配置 qmake

➢ 配置 C++编译器

我们在 1.3 小节安装的 C++编译器为 arm-linux-gnueabihf-g++。按如下步骤配置，选择 Compiler path 的路径为我们 1.2 小节安装的交叉编译器路径/usr/local/arm/gcc-linaro-4.9.4-2017.01-x86_64_arm-linux-gnueabihf/bin/arm-linux-gnueabihf-g++。并在第 13 步修改 Name 为 ARM-GCC。

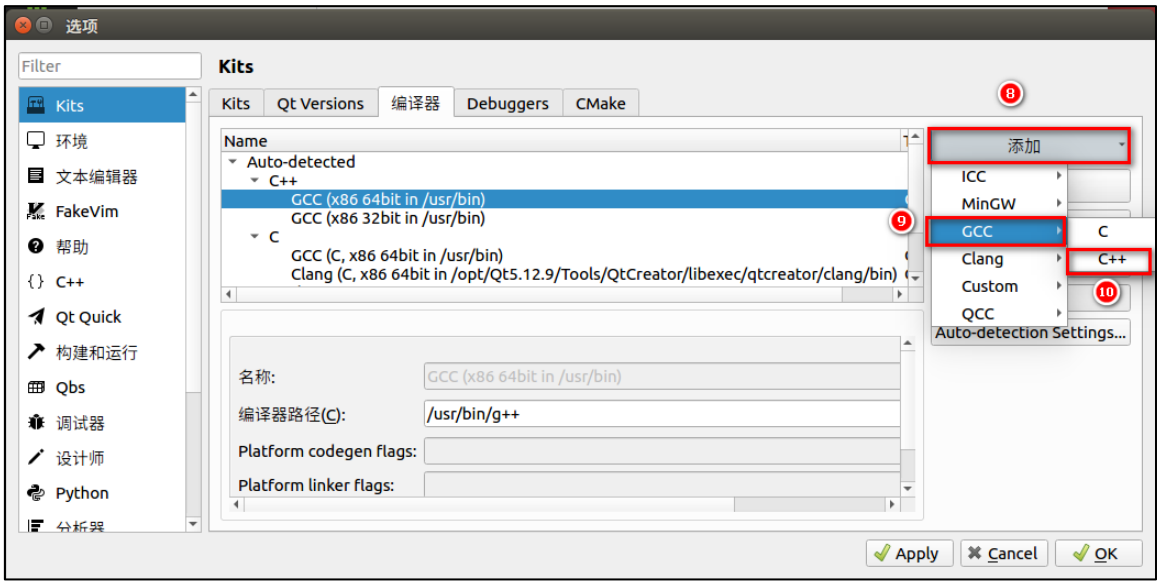


图 5.2 4 配置 GCC

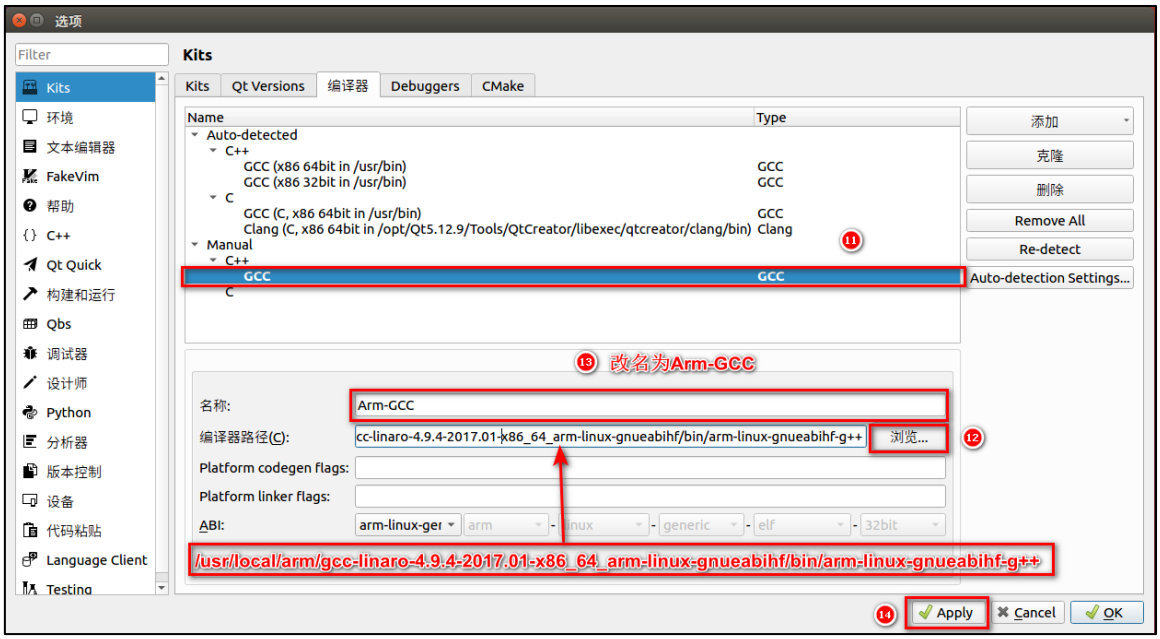


图 5.2 5 配置 GCC 完成

➢ 配置 Kits

按如下图步骤操作，在 Name 处改名为 Arm-Qt5.12.9。选择我们配置好的 Compiler 和 Qt Version 即可。

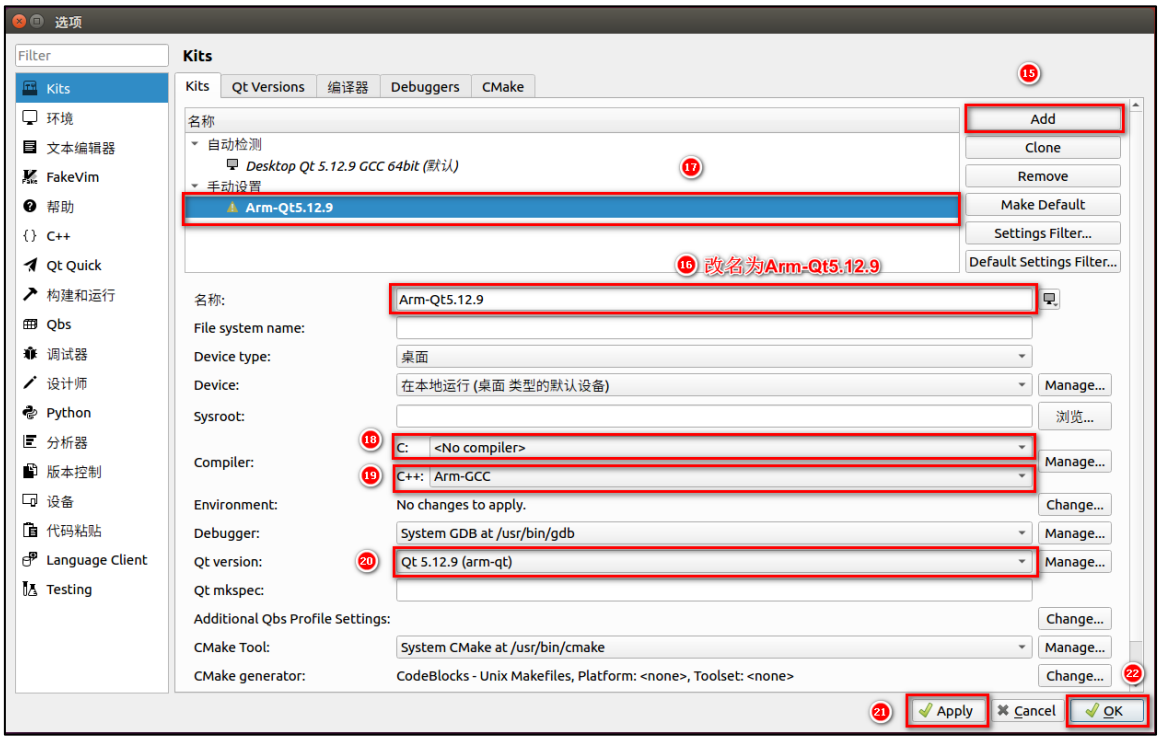


图 5.2 6 配置 Kits

5.3 验证 ARM 平台的 Qt 编译

注意，这里只能编译(构建)，不能点击运行，这里是编译出 ARM 平台的 Qt 应用程序，所以是不能在 Ubuntu 这样的 X86 平台上跑的。

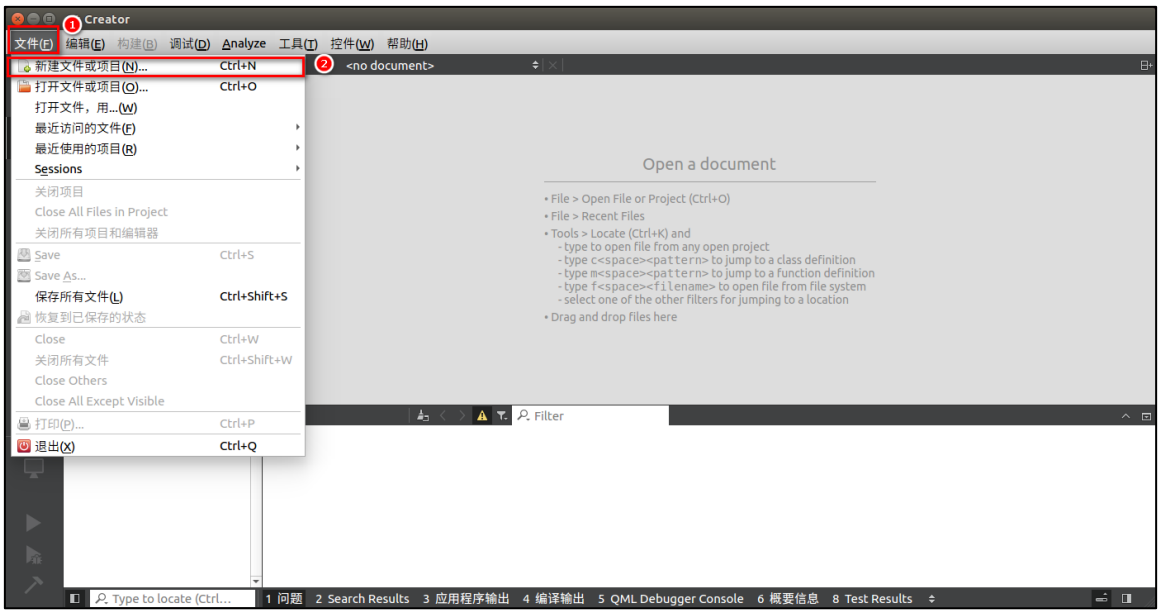


图 5.3 1 新建项目

选择 Application 项目和 Qt Widgets Application 模板。

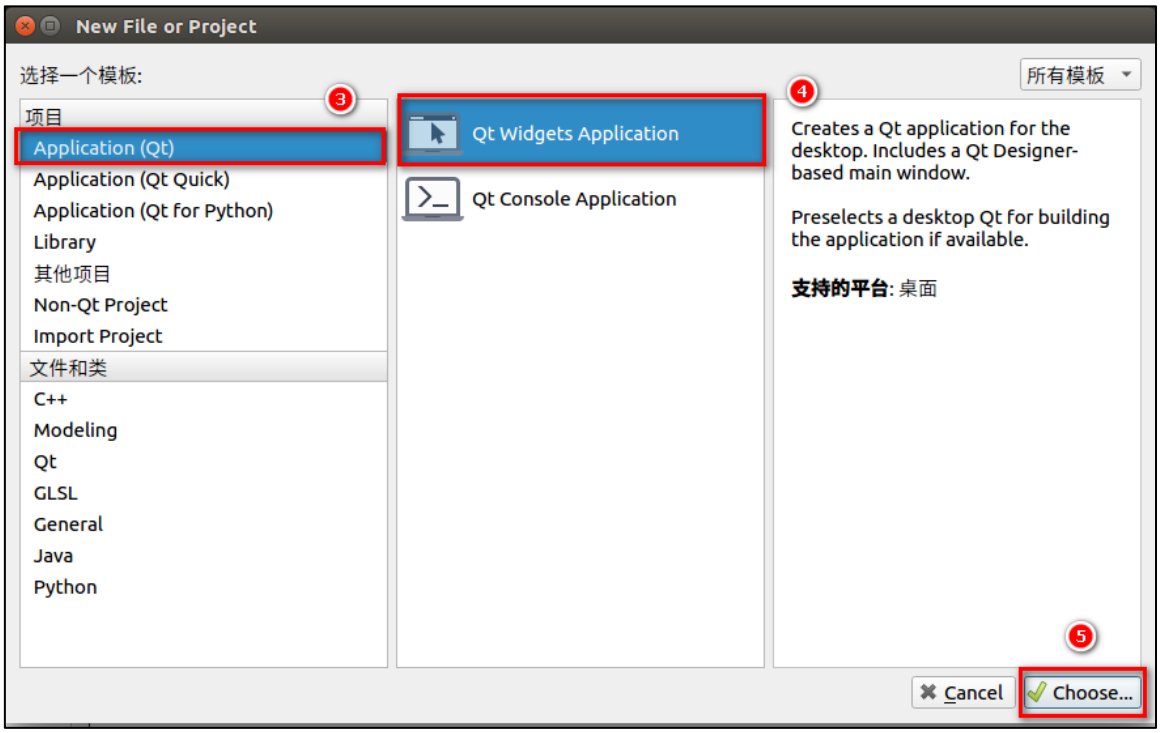


图 5.3 2 选择项目与模板

项目命名为 test，选择工程的位置，这里位置不要随便选择，建议放在家目录下（/home/用户名）。

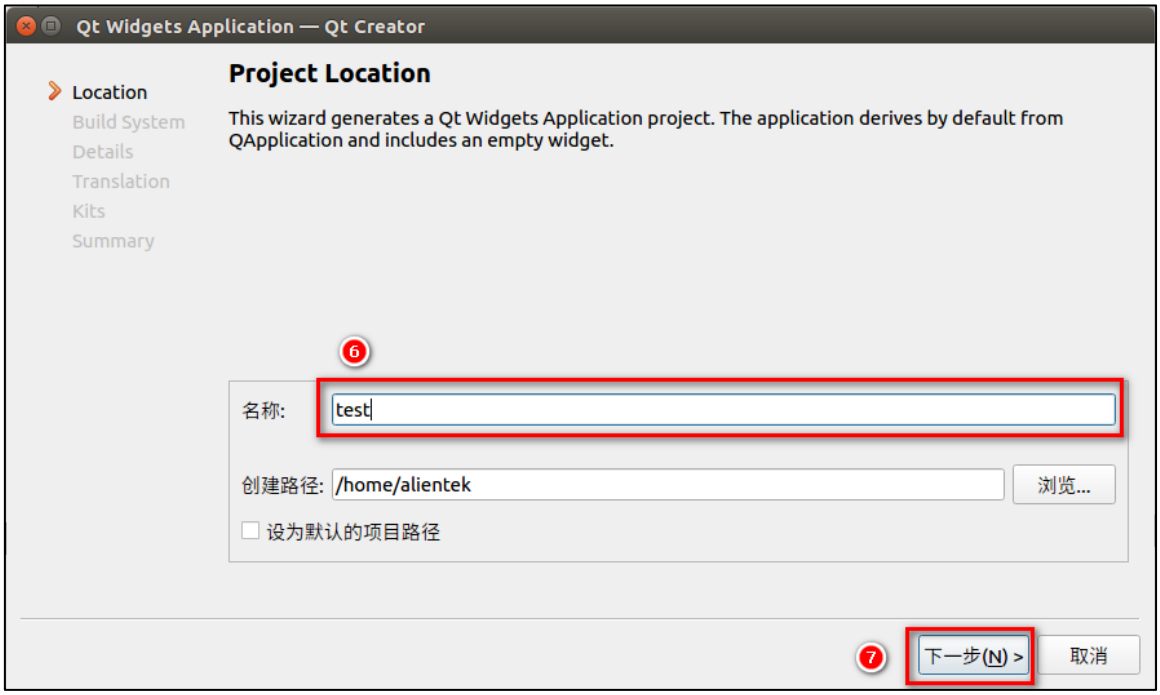


图 5.3 3 项目命名为 test

默认使用 qmake，点击下一步。

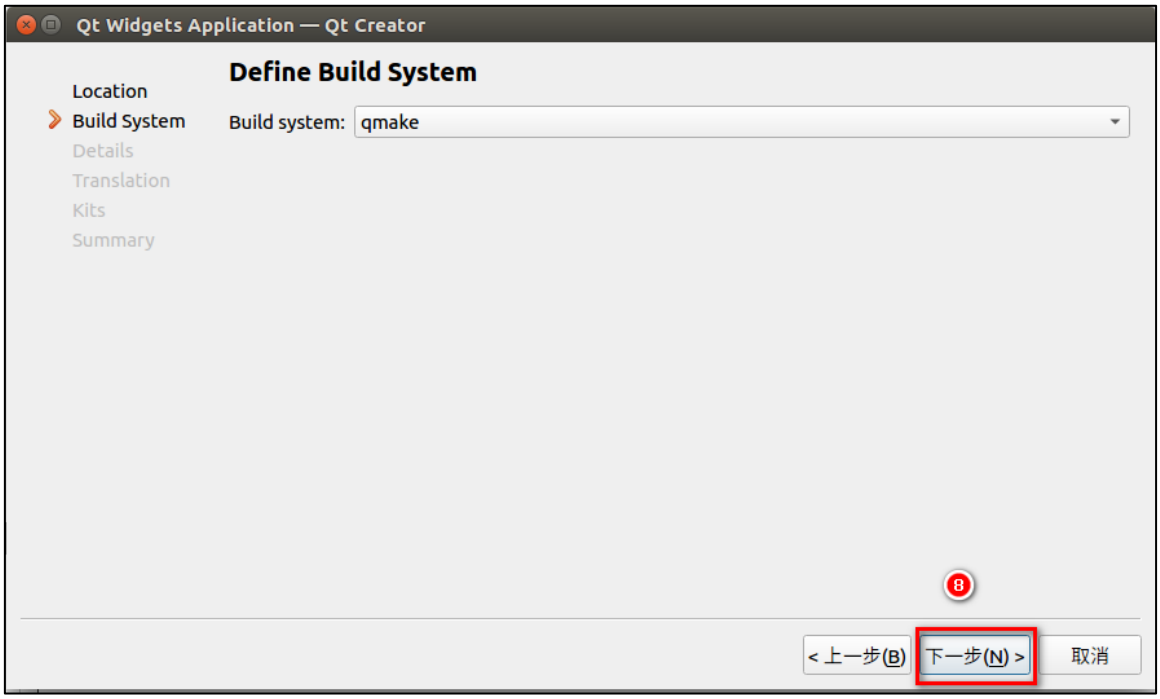


图 5.3 4 默认构建系统

选择类模板，默认 QMainWindow 类即可。

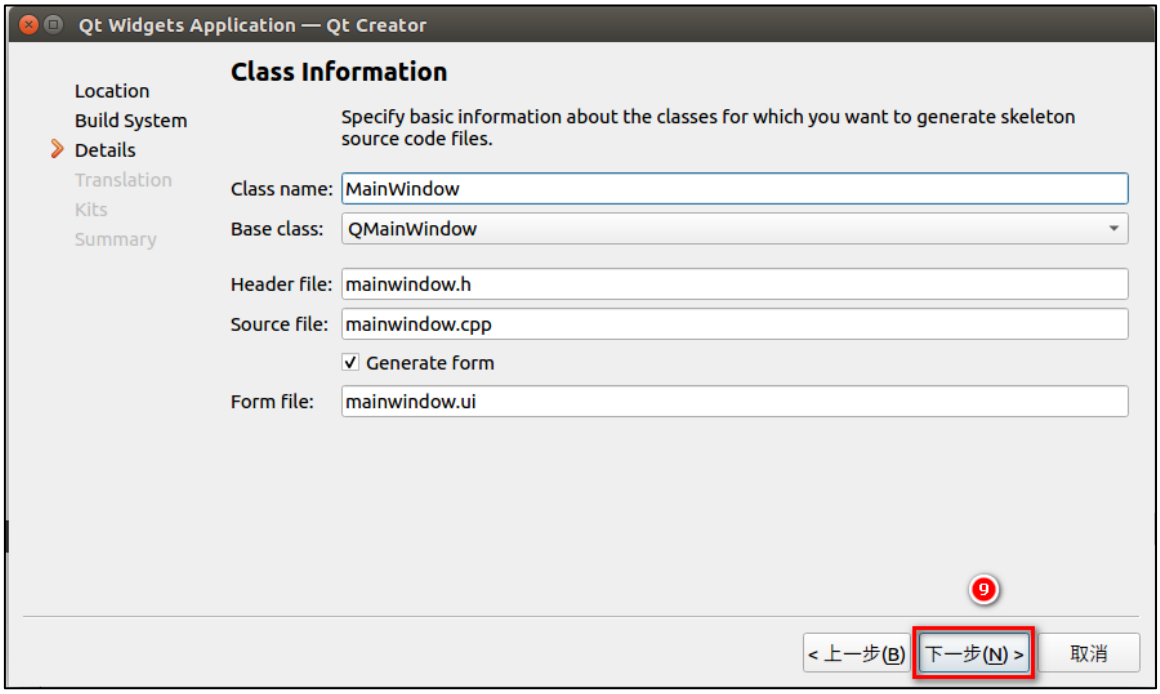


图 5.3 5 选择类模板

选择文件翻译，默认无，点击下一步。

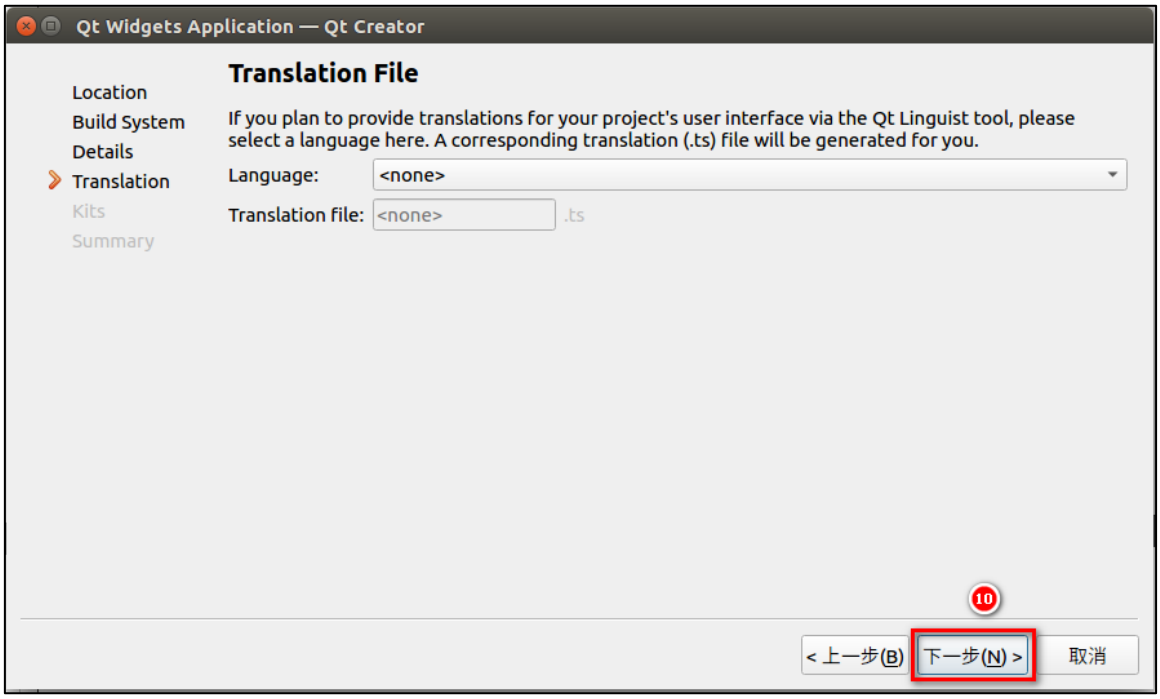


图 5.3 6 跳过文件翻译

选择套件，这里可以两个套件一起选，编译时切换选择 ARM 平台的套件，如下。

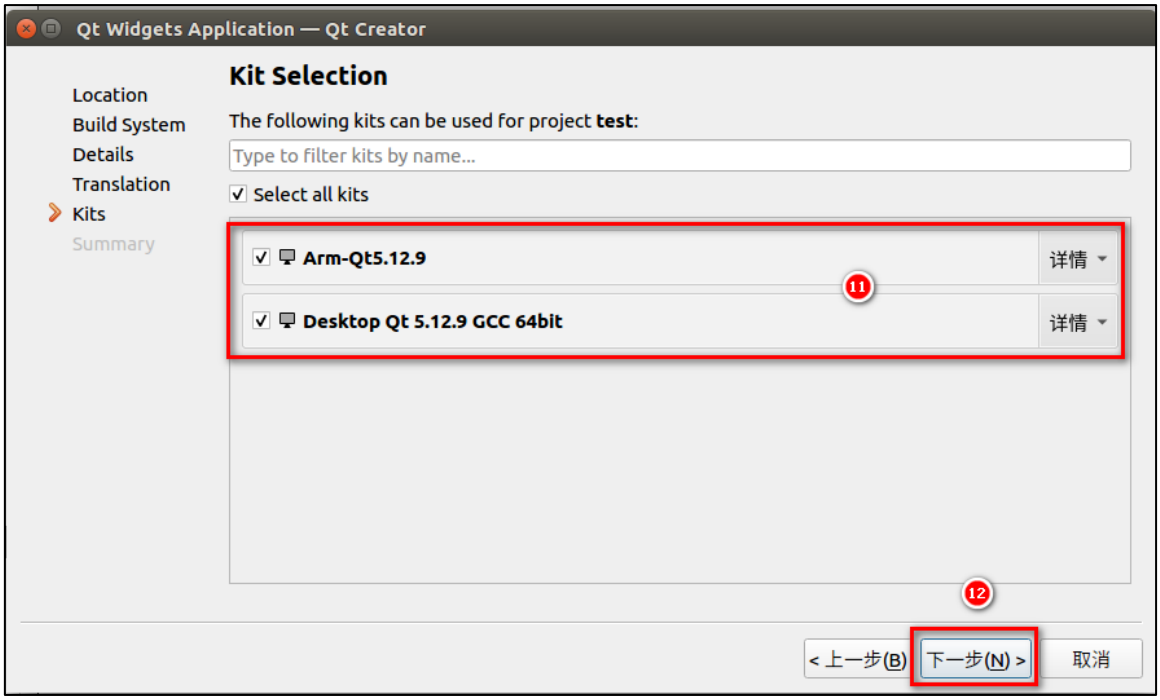


图 5.3 7 选择 Kits

默认，点击 Finish，完成。

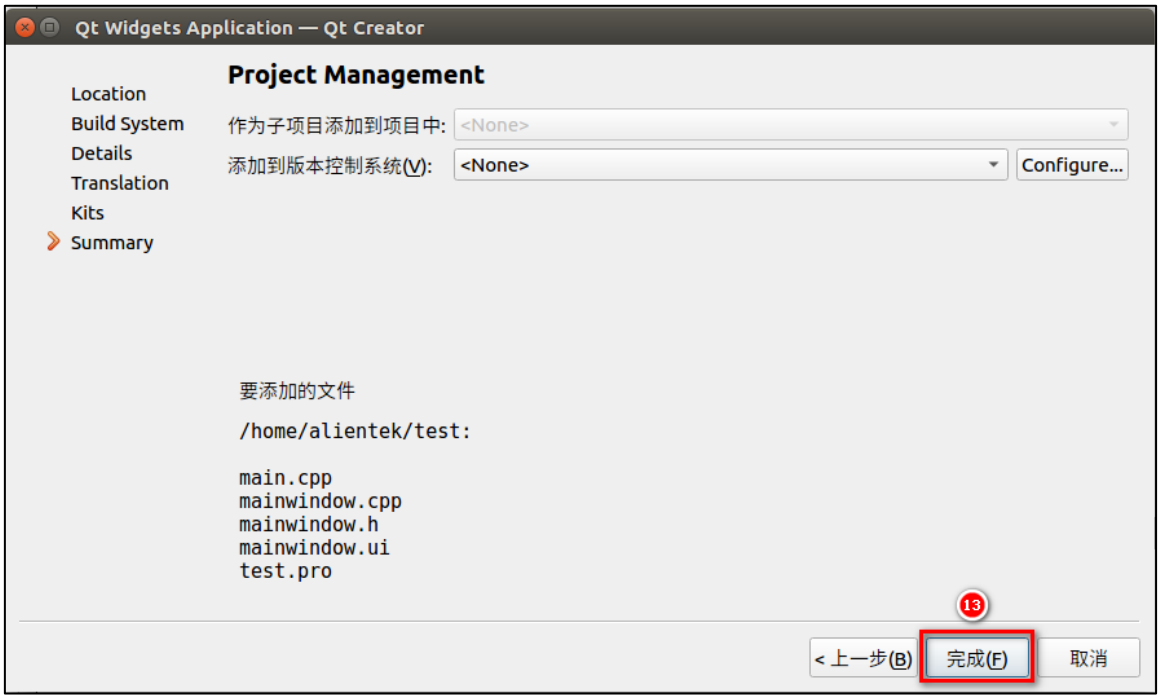


图 5.3 8 跳过版本控制

选择 ARM 平台所用的 Kits。选择 Debug 构建。

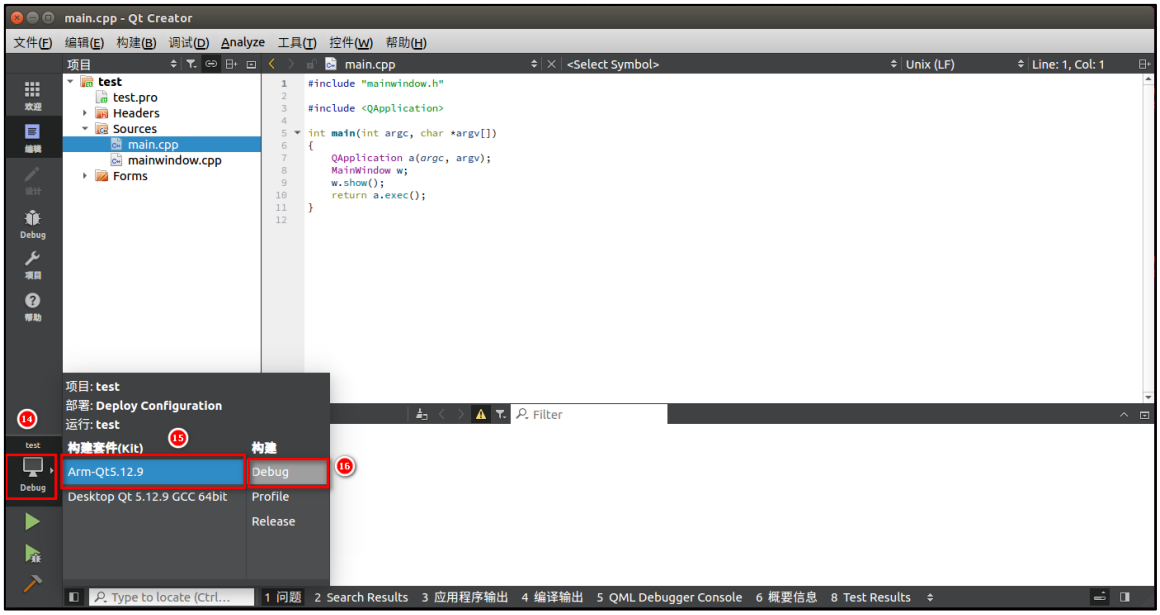


图 5.3 9 选择 Kits 构建

按如下步骤点击 Build（构建），不要点击运行！也不要点击左下角的绿色三角符号。编译完成如下图，编译输出窗口。

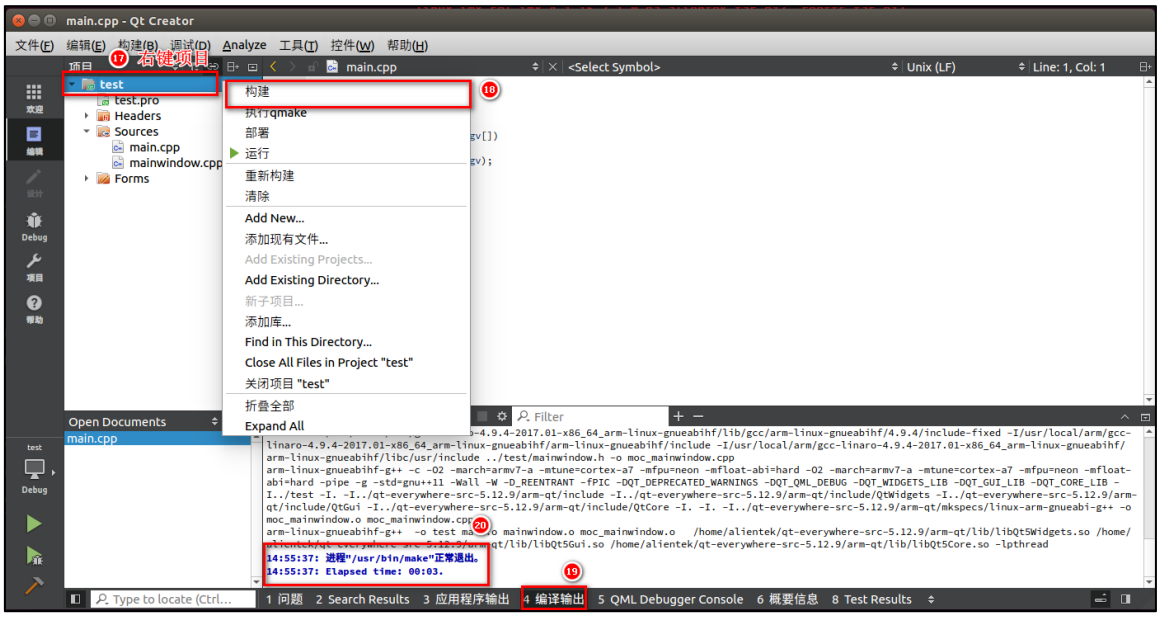


图 5.3 10 点击构建，查看编译输出窗口

我们可以在工程目录所在的同一级目录下，找到 build-test-Arm_Qt5_12_9-Debug/文件夹，找到编译可执行文件，拷贝 test 文件到 4.1 小节开发板文件系统下上执行即可！执行命令为./test。

图 5.3 11 查看编译输出文件

5.4 命令行编译 Qt 工程

前提在第 1.3 小节已经安装了交叉编译器，然后我们可以直接在 test 工程文件夹里，直接运行 qmake，生成 Makefile，再执行 make 开始编译。

```
cd test                    // 进入 test 工程文件夹下
/home/alientek/qt-everywhere-src-5.12.9/arm-qt/bin/qmake  // qmake 的绝对路径，需要改为自己的
make -j 16
```

编译完成如下图，会在当前文件夹生成 test 文件，拷贝到 4.1 小节开发板文件系统下直接使用
./test 执行即可。

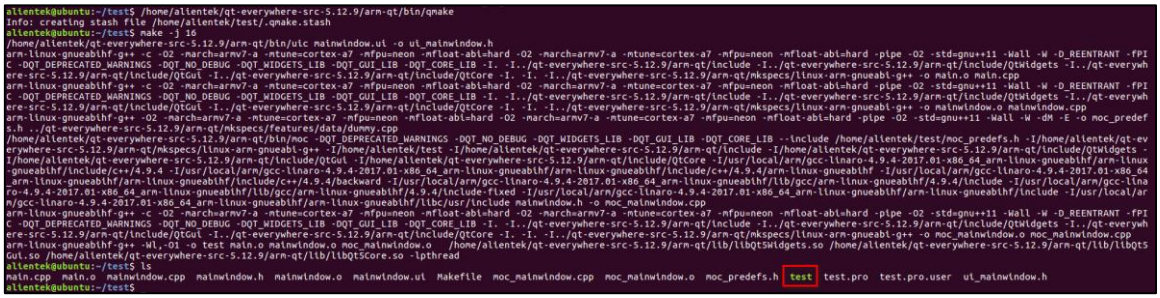


图 5.4 1 命令行编译 Qt 工程

至此，已经完成关于 Qt5.12.9 的移植及 Qt Creator 环境搭建。希望对大家有所帮助。

附录-A