技 术 文 件

文件名称：Git代码管理指南

文件编号：

版 本：

|  |  |
| --- | --- |
| 拟 制 | Hardy |
| 审 核 |  |
| 批 准 |  |

修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文件编号 | 版本号 | 拟制人/  修改人 | 拟制日期/  修改日期 | 更改理由 | 主要更改内容  （写要点即可） |
|  | V1.0 | Hardy | 2017/11/27 | 首版。 | 文档框架。 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| 注：文件第一次归档时，“更改理由”、“主要更改内容”栏写“无”。 | | | | | |

目 录

[1 引言 1](#_Toc512521300)

[1.1 编写目的 1](#_Toc512521301)

[1.2 文档约定 1](#_Toc512521302)

[2 术语、定义和缩略语 1](#_Toc512521303)

[2.1 术语、定义 1](#_Toc512521304)

[2.2 缩略语 1](#_Toc512521305)

[3 相关文档 2](#_Toc512521306)

[4 编译环境 2](#_Toc512521307)

[4.1 直接在测试仪上开发 2](#_Toc512521308)

[4.2 在PC主机上交叉编译 2](#_Toc512521309)

[4.2.1 主机环境搭建 3](#_Toc512521310)

[4.3 程序部署及数据共享 3](#_Toc512521311)

[4.3.1 U盘拷贝 3](#_Toc512521312)

[4.3.2 Samba服务器 3](#_Toc512521313)

[4.3.3 FTP 3](#_Toc512521314)

[5 项目开发 4](#_Toc512521315)

[5.1 Makefile方式 4](#_Toc512521316)

[5.1.1 动态库生成模板 4](#_Toc512521317)

[5.1.2 可执行程序生成模板 4](#_Toc512521318)

[5.2 CMake 4](#_Toc512521319)

[5.3 使用QT的软件工具包 4](#_Toc512521320)

[5.3.1 工程建立 4](#_Toc512521321)

[5.3.2 Windows平台 8](#_Toc512521322)

[5.3.3 测试仪Linux平台 8](#_Toc512521323)

[5.4 Linux平台常用调试手段 8](#_Toc512521324)

[5.4.1 日志系统 8](#_Toc512521325)

[5.4.2 gdb调试工具 8](#_Toc512521326)

[6 测试仪平台软件性能优化 8](#_Toc512521327)

[6.1 CUDA编程 8](#_Toc512521328)

[6.2 NEON指令优化 8](#_Toc512521329)

[7 参考资料 8](#_Toc512521330)

# 引言

## 编写目的

本文档描述了使用GIT进行代码管理的基本操作方法。

## 文档约定

本文档遵循以下约定：

1. 使用的模板是：“技术文档模板V1.0”。
2. 文档的密级分为：机密、秘密、内部公开、公开发布。
3. 表头文字使用了20%灰度背景。
4. 插图一律“嵌入”于描述正文中，而非“浮于文字上方”。
5. 用同号、同体但加粗的文字来强调需要读者重视的内容。
6. 表格和图片统一使用“题注”，编号方式采用“包含章节号”，标签方式采用“图表”，表格的题注放于表格的上方，图片的题注放于图片的下方，其它选项使用默认值。
7. 文中引用题注时，统一使用“交叉引用”，引用内容“只有标签和编号”。

# 术语、定义和缩略语

## 术语、定义

本文使用的专用术语、定义见表 1。

表 1 术语、定义

| 术语/定义 | 英文 | 说 明 |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

## 缩略语

本文使用的专用缩略语见表 2。缩略语按其第1个字母顺序排列。

表 2 缩略语

| 缩略语 | 英文原文 | 中文含义 |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# 相关文档

本文涉及的相关文档表 3

表 3 相关文档

| 文件编号 | 文件名称 | 版本号 | 说明 |
| --- | --- | --- | --- |
|  |  |  | 上游文档 |
|  |  |  | 上游文档 |
|  |  |  | 引用文档 |
|  |  |  | 引用文档 |
|  |  |  | 引用文档 |
|  |  |  | 引用文档 |
|  |  |  | 引用文档 |

# 概念介绍

辰卓科技模组测试仪是使用与模组终端平台一致的硬件平台，其硬件架构遵循冯诺依曼计算机的组织结构，包括基于CPU+GPU的SOC处理器、存储器、输入输出设备。软件系统是基于Ubuntu的Linux操作系统，包括了驱动、内核、文件系统及应用程序等部分。

在测试仪上进行软件二次开发，主要是编译出能运行在ARM Linux平台下的二进制目标代码，主要包括可执行程序和动态库。

## 直接在测试仪上开发

## 添加密钥

命令：$ssh-keygen -t rsa -C "youremail@example.com"。

你需要把邮件地址换成你自己的邮件地址，然后一路回车，使用默认值即可，

登陆GitHub，打开“Account settings”，“SSH Keys”页面：

然后，点“Add SSH Key”，填上任意Title，在Key文本框里粘贴id\_rsa.pub文件的内容：

github-addkey-1

点“Add Key”，你就应该看到已经添加的Key：

github-addkey-2

为什么GitHub需要SSH Key呢？因为GitHub需要识别出你推送的提交确实是你推送的，而不是别人冒充的，而Git支持SSH协议，所以，GitHub只要知道了你的公钥，就可以确认只有你自己才能推送

### 生成SSH-Key

1. 打开Git Bash，输入以下命令，替换成你自己的email地址：

$ ssh-keygen -t rsa -b 4096 -C "your\_email@example.com"

此命令将使用新提供的email创建一个新的SSH Key。

Generating public/private rsa key pair.

当提示"Enter file in which to save the key"时按回车键，表示接受默认文件路径。

Enter a file in which to save the key (/c/Users/you/.ssh/id\_rsa):[Press enter]

输入两次密钥。

Enter passphrase (empty for no passphrase): [Type a passphrase]

Enter same passphrase again: [Type passphrase again]

由于这个Key也不是用于军事目的，所以也无需设置密码。

如果一切顺利的话，可以在用户主目录里找到.ssh目录，里面有id\_rsa和id\_rsa.pub两个文件，这两个就是SSH Key的秘钥对，id\_rsa是私钥，不能泄露出去，id\_rsa.pub是公钥，可以放心地告诉任何人。

### 主机环境搭建

通常情况下，主机就是一台装有任意Linux发行版本的PC电脑。这里，我们以基于x64硬件架构和Ubuntu发行版的Linux操作系统电脑为例。

我们的Ubuntu操作系统可以是直接安装在物理PC机上。如果我们是使用的Windows操作系统，则需要安装双系统或在虚拟机中安装Ubuntu系统。

下面假设我们有一台Windows 7操作系统的PC机作为程序开发的主机。若物理机上有Linux系统，则略过前两节。

#### 安装虚拟机

虚拟机软件有很多种，常用的有VMWare或Virtual Box。

#### 安装Ubuntu操作系统

装好虚拟机后，就可以在虚拟机中安装Ubuntu操作系统了。Ubuntu操作系统当前推荐使用Ubuntu 16.04 LTS。

#### 安装交叉编译器

交叉编译器使用gcc-linaro-5.4.1-2017.01-x86\_64\_aarch64-linux-gnu。这里选用的是2017年1月发布的基于gcc5.4.1的版本，理论上如果没有用新版本编译器的新特性，应该是向下兼容的，如4.8.2版本。

官方下载链接为：

x64位环境：

<http://releases.linaro.org/components/toolchain/binaries/5.4-2017.01/aarch64-linux-gnu/gcc-linaro-5.4.1-2017.01-x86_64_aarch64-linux-gnu.tar.xz>

x32位环境：

<http://releases.linaro.org/components/toolchain/binaries/5.4-2017.01/aarch64-linux-gnu/gcc-linaro-5.4.1-2017.01-i686_aarch64-linux-gnu.tar.xz>

下载完成后，这是一个软件压缩包。解压后即可以使用。

$ tar Jxvf \*.tar.xz

## 程序部署及数据共享

目标机程序部署或数据共享有很多种方式。

### U盘拷贝

最简单的就是使用U盘拷贝程序或其他文件到目标机中，可以在图形用户界面操作，也可以通过终端命令cp拷贝。

### Samba服务器

Samba服务是Linux提供文件共享的常用服务，测试仪预装了些服务，其配置文件是/etc/samba/smb.conf，可以根据需要修改或使用默认配置。服务启动成功后，在Windows下直接输入测试仪的IP即可以发现共享的文件夹进行数据交换。

### FTP

FTP也是一种常用的数据传输服务，如果需要，用户可以自行配置。

## 参考链接

<https://help.github.com/articles/generating-a-new-ssh-key-and-adding-it-to-the-ssh-agent/>

# GitHub管理

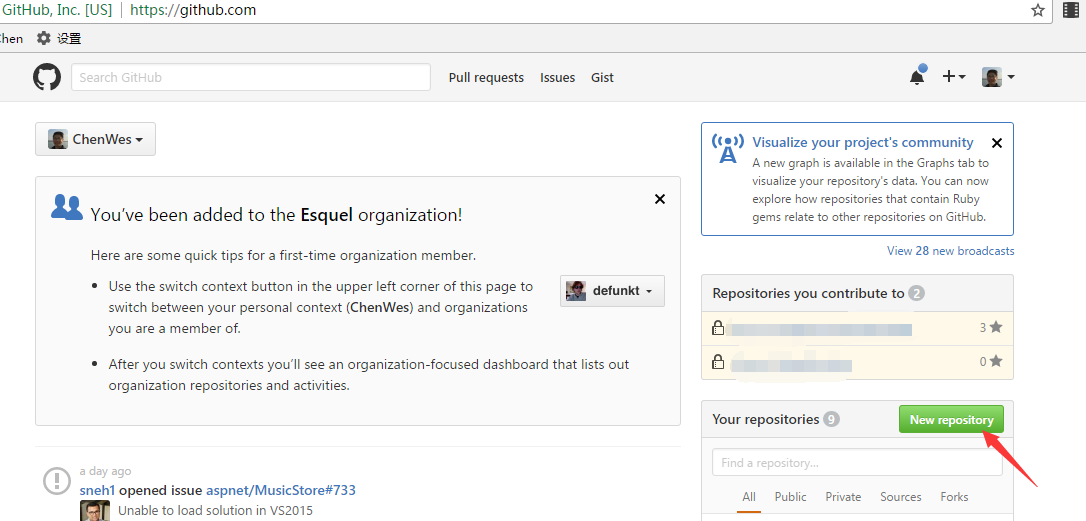
GitHub是当前开源项目托管比较多的一个境外Git服务器网站，创建公开的仓库是免费的，私有的仓库是收费的。

## 创建仓库流程

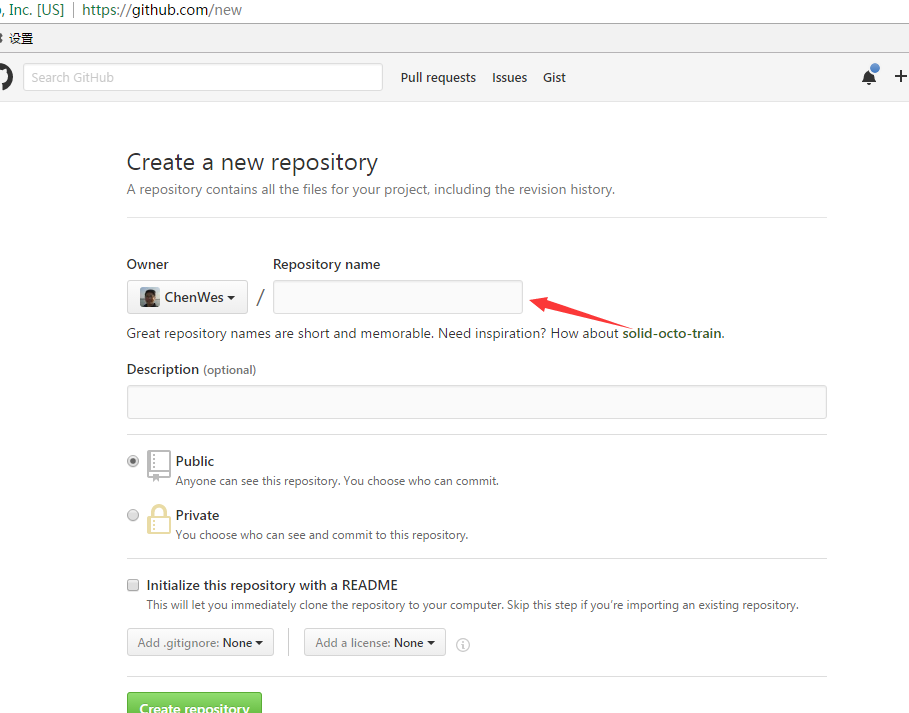
### 创建Github代码仓库

打开www.github.com，创建一个帐号（如是公司，应该创建一个组织）

在下图中，点击new repository可以开始创建一个新的仓库



在创建仓库时，填写项目名称，这里，我以个人为例，这里的代码仓库是公开的

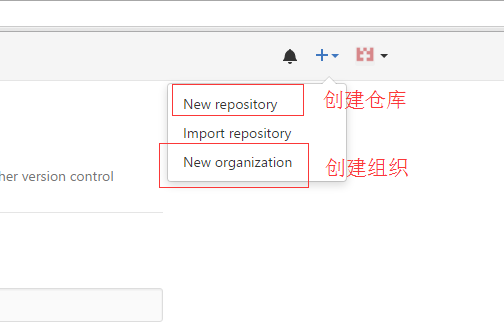


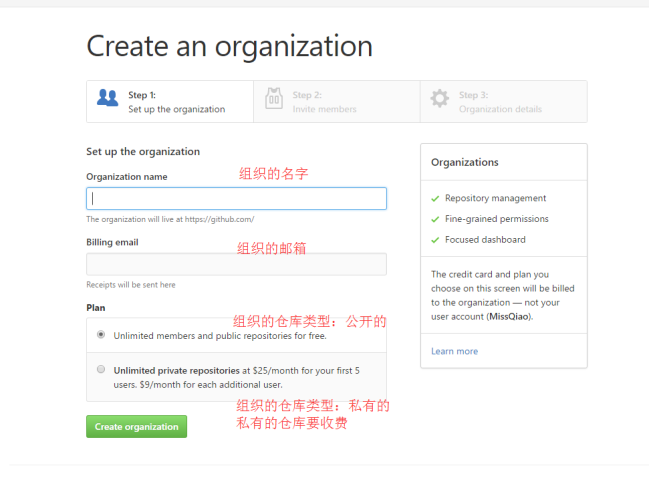
### 权限管理

要想弄清楚github权限管理，需要了解几个概念。 **organization（组织） 、repository（仓库）、Team（团队）**。

#### 创建组织

github上的用户可以创建仓库和组织。





#### 在组织中创建仓库

当创建完组织后，来看看组织的结构



在Organ-Name 组织下，创建一个仓库

#### 在组织中创建team

团队创建完成后，默认这个团队的成员只有一个人，就是该账号。下面就开始给这个team添加其他成员。

邀请成功以后，需要被邀请人去自己的邮箱点击一下，确认邀请

#### 在组织中给仓库添加team并且设置权限

可以看到，仓库对team的权限控制有三种

Admin 管理者权限(只能 read、clone、push、给仓库添加成员 )

Write 写权限(只能 read、clone、push)

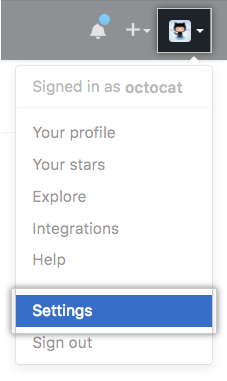
Read 读权限(只能 read、clone)

另外任意一个Team可以供多个组织使用，到这里权限添加已经全部完成了。

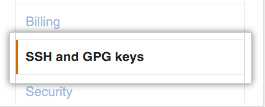
## 添加SSH-Key

拷贝SSH Key公钥（如id\_rsa.pub）到剪切板。

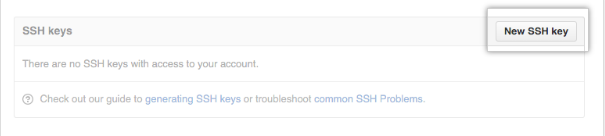
In the upper-right corner of any page, click your profile photo, then click **Settings**.



In the user settings sidebar, click **SSH and GPG keys**.

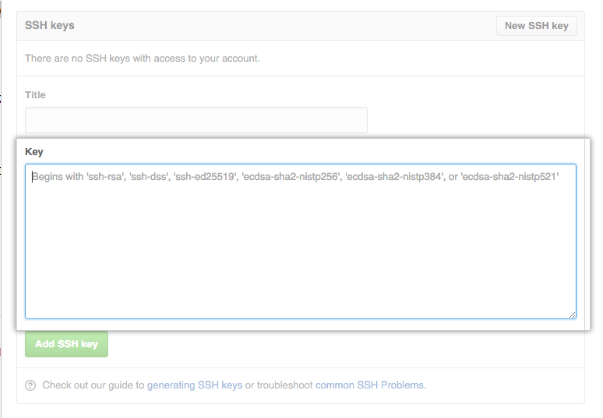


Click **New SSH key** or **Add SSH key**.

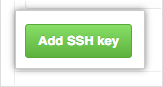


In the "Title" field, add a descriptive label for the new key. For example, if you're using a personal Mac, you might call this key "Personal MacBook Air".

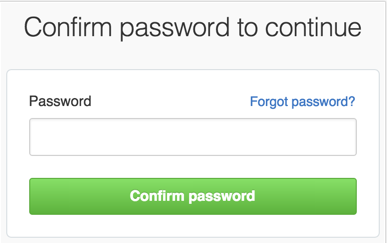
Paste your key into the "Key" field.



Click **Add SSH key**.



If prompted, confirm your GitHub password.



## 小结

github不能免费的创建私有仓库，这是一个很严重的问题。如果是开源项目，用github完全没有问题。如果是私有项目，可以有以下几个途径达到要求

1、在github花钱购买私有仓库。

2、使用国内比较出名的开源中国git托管服务：https://git.oschina.net/

3、使用GitLab,这需要在自己的服务器上部署。传送门：https://about.gitlab.com/gitlab-com/

## 参考链接

使用GitHub管理源代码

<https://www.cnblogs.com/weschen/p/6264056.html>

GitHub 实现多人协同提交代码并且权限分组管理

<https://www.cnblogs.com/zhaoyanjun/p/5882784.html>

## Makefile方式

最手工最纯粹的方式。手工组织目录结构将代码分模块存储，然后写一个Makefile文件来管理这些代码文件，并在其中调用gcc编译器来生成目标代码。

以下是两个Makefile文件模板：

### 动态库生成模板

### 可执行程序生成模板

## CMake

CMake可以用来管理项目工程，同时能自动生成Makefile文件，从而编译生成目标代码。

## 使用QT的软件工具包

这是一种较为简单且推荐的方法。

QT是一种跨平台的C++开发框架，含有丰富的类库，能写各种常见的应用程序、动态库、静态库、插件等。

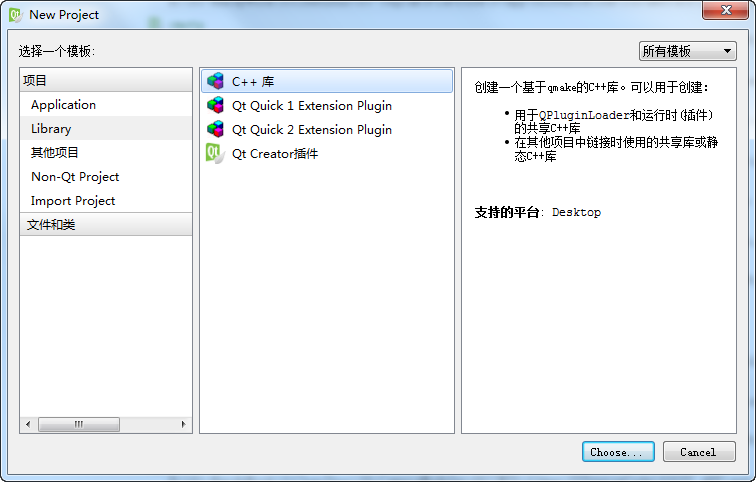
我们在开发过程中，可以使用QT自身的类库，也可以选择不使用QT的类库而用标准C++库来完成编程

由于QT的跨平台性，其集成开发环境能正确识别Windows上创建的工程，我们完全可以在Windows上完成整个程序开发，然后将代码在Linux下重新编译即能在Linux平台使用。

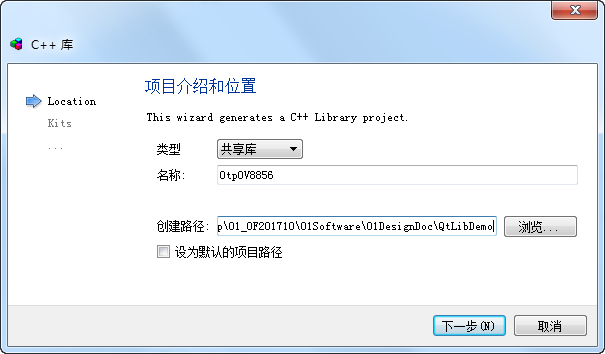
下面以开发一个算法动态库为例，讲解在Windows和Linux下，QT工程的使用方法。

### 工程建立

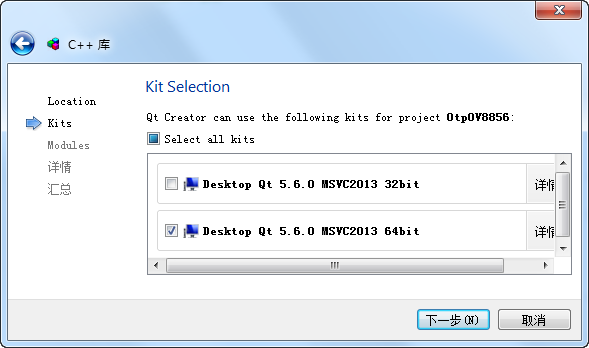
新建一个C++库工程。



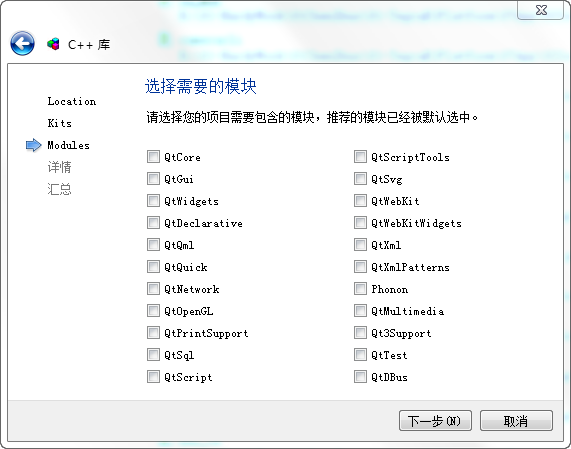
创建动态库。



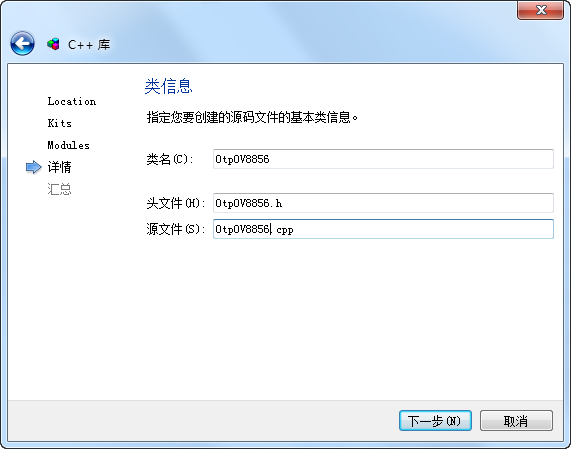
选择版本和编译模型。



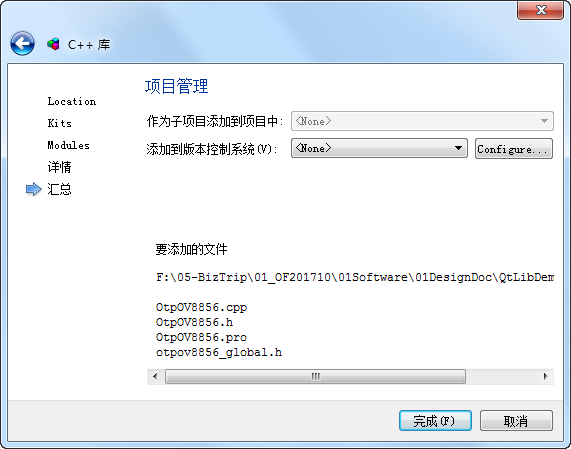
选择需要的模块。所有模块均不选中，即不使用QT类库。



类信息。保存文件时，QT默认全小写，建议改为和类名一致。



完成工程创建。



对自动生成的.pro工程文件稍作修改，具体修改方法可参考QT帮助文档。修改后的主要内容大致如下：

QT -= core gui

CONFIG += c++11 plugin

TARGET = OtpOV8856

TEMPLATE = lib

CONFIG += debug\_and\_release

CONFIG(debug, debug|release) {

DIST\_DIR = Debug

} else {

DIST\_DIR = Release

QMAKE\_POST\_LINK = $(STRIP) $(DESTDIR)$(TARGET)

}

contains(QT\_ARCH, i386) {

message("32-bit")

CPU\_ARCH = x86

} else {

message("64-bit")

CPU\_ARCH = x64

}

DESTDIR = $$PWD/../lib/$${CPU\_ARCH}/$${DIST\_DIR}

DEFINES += OTP\_EXPORTS

SOURCES += OtpOV8856.cpp

HEADERS += OtpOV8856.h

INCLUDEPATH += $$PWD/../includes/

unix {

target.path = /usr/lib

INSTALLS += target

}

主要修改了输出目录及引用头文件搜索目录。CONFIG增加了c++11和plugin特性。

删除默认的otpov8856\_global.h文件，此文件主要是导出符号的定义。

### Windows平台

上节中自动生成的代码可直接编译。在Windows上可以直接在IDE上选择菜单“构建->执行qmake”，此步骤会生成Makefile文件。再选择菜单“构建->构建项目xxx”，即可编译生成目标代码。

### 测试仪Linux平台

测试仪Linux平台上如果使用QCreator图形用户界面，操作方式与Windows一样。

通常在Windows下编辑好的代码，不需要在Linux下再编辑，这时使用终端用户界面就够了，将源代码拷贝到测试仪某个用户目录下。如果使用Samba服务建立了目录共享，还能使用BeyondCompare等工具进行代码实时比对。

在终端用户界面中，使用cd命令进行源代码所在目录。执行qt的qmake命令可以将qt的\*.pro工程生成Makefile文件：

$ qmake OtpOV8856.pro

再根据生成的Makefile文件执行相应的make命令：

$ make -j4

然后根据编译器给出的编译提示信息修改错误或调试代码。

## Linux平台常用调试手段

### 日志系统

在程序中建立功能完善的日志系统，有利于分析代码的执行流程，当程序执行异常时，可通过日志文件来分析代码是否有报错来判断BUG。

### gdb调试工具

gdb是Linux系统下功能非常完善的在线调试工具。其支持的命令很多，具体可以查看帮助文件。例如使用backtrace命令来调出程序崩溃时的调用栈。

# 测试仪平台软件性能优化

## CUDA编程

测试仪平台支持CUDA编程，利用GPU的并行计算，大大优化算法性能。

## NEON指令优化

# 参考资料