V1.2
2026/09/30

H2 API

范例使用指南

HAROGIC 海得逻捷

目录

版本管理 1

1. Windows 系统 2

1.1 C/C++ 2
1.1.1 配置开发环境 2
1.1.2 范例使用流程 4
1.2 Qt 4
1.2.1 配置开发环境 4
1.2.2 范例使用流程 6
1.3 Python 7
1.3.1 配置开发环境 7
1.3.2 范例使用流程 8
1.4 C# 9
1.4.1 配置开发环境 9
1.4.2 范例使用流程 11
1.5 Matlab 12
1.5.1 范例使用流程 12

2. Linux 系统 14

2.1 系统运行环境检查 14
2.2 随寄资料说明 15
2.2.1 C++范例 15
2.2.2 Qt 范例 15
2.2.3 Python 范例 15
2.2.4 Install_H2_SDK 16
2.3 配置驱动文件 17
2.4 C/C++ 18
2.4.1 范例使用流程 18
2.4.2 创建与编译新项目 19
2.4.3 交叉编译 22
2.5 Qt 23
2.5.1 范例使用流程 23
2.5.2 创建与编译新项目 25
2.5.3 交叉编译 28
2.6 Python 31
2.6.1 范例使用流程 31
2.6.2 创建与编译新项目 32

3. 范例说明 33

3.1 发射连续波 33

3.2 回放.....33
3.3 流.....33
3.4 动态更改配置.....33
3.5 频率扫描.....34
3.6 功率扫描.....34
3.7 调制.....34

版本管理

版本更新说明表

版本号	内容	时间
V1.2	1. 修改：范例说明章节	2026-09-30
V1.1	1. 增加：Windows 系统中新增 Matlab 范例使用说明 2. 增加：Linux 系统下范例使用说明	2026-05-21
V1.0	1. 初始版本	2026-03-19

1. Windows 系统

1.1 C/C++

1.1.1 配置开发环境

下文以配置 Debug Win32 为例，若需配置其他构建类型和平台，请参考本章文末。

1. 打开 Visual Studio 2019，创建一个新项目（例如 test）；

2. 创建完成后，将发货 U 盘“Windows\api”文件夹拷贝到工程的平级目录下；

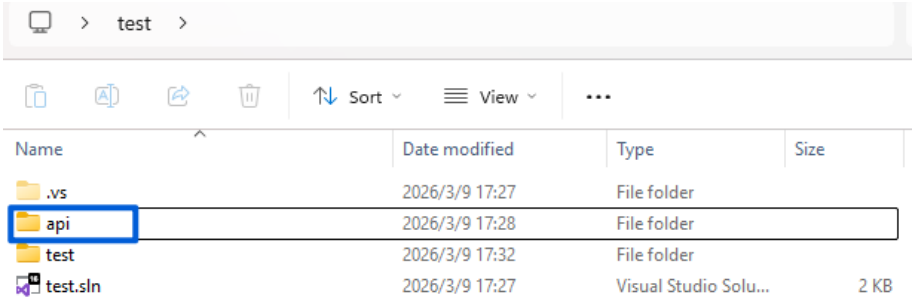


图 1 拷贝 api 文件

3. 进入 Visual Studio 2019，右击“源文件”->“添加”->“新建项”->“C++文件(.cpp)”，新建 main.cpp 文件；

4. 点击菜单栏中的“项目”->“属性”，将“配置”设置为“Debug”，“平台”设置为“Win32”；

5. 将 配置属性 -> 调试中环境变量设置为“Path=..\api\x86”；

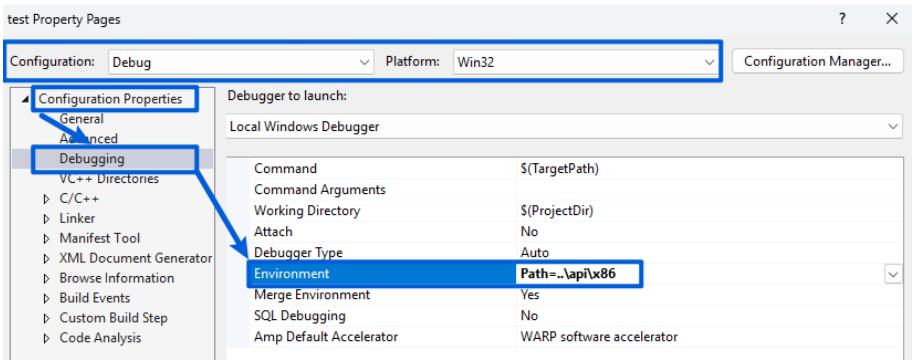


图 2 配置环境变量

6. 将配置属性 -> C/C++ -> 常规中的附加包含目录设置为“$(SolutionDir)\api\x86”；

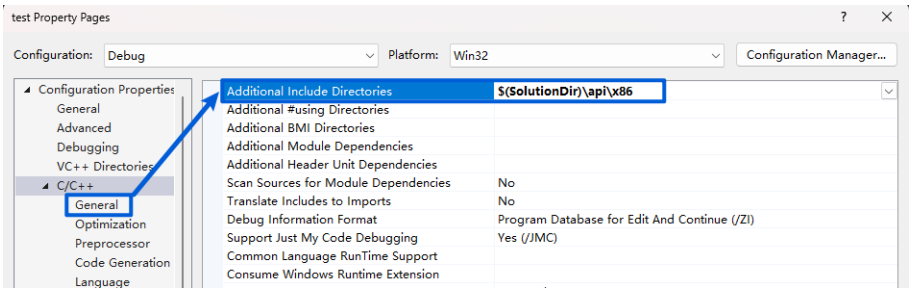


图 3 添加附加包含目录

7. 将配置属性 -> 链接器 -> 常规 中的 附加库目录 设置为“$(SolutionDir)\api\x86”；

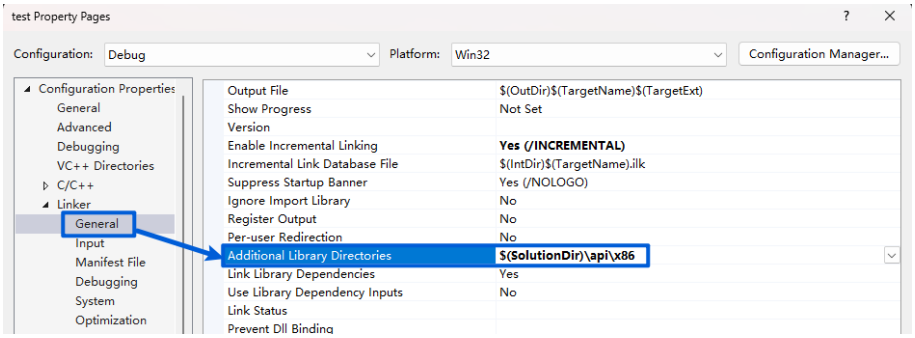


图 4 配置附加库目录

8. 给配置属性 -> 链接器 -> 输入 中的“附加依赖项”新增“h2_api.lib”，并点击“OK”；

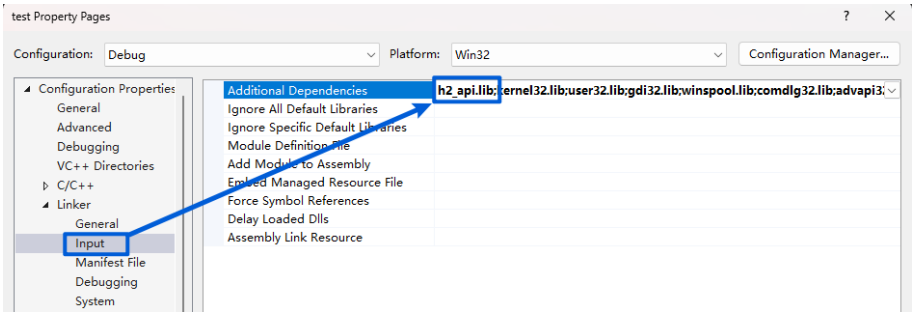


图 5 配置附加依赖

- 配置 Release Win32：将步骤 4 中的“配置”设置为“Release”，其余保持一致。
- 配置 Debug x64：将步骤 4 中“配置”设置为“Debug”，“平台”设置为“x64”；同时将所有环境变量或者附加目录中“api\x86”替换为“api\x64”。
- 配置 Release x64：将步骤 4 中“配置”设置为“Release”，“平台”设置为“x64”；同时将所有环境变量或者附加目录中“api\x86”替换为“api\x64”。

1.1.2 范例使用流程

注：同一时间只能运行一个范例，范例与软件不可同时运行。

1. 使用 Visual Studio 2019 或更高版本软件打开随寄 U 盘“Windows\example\C++”文件夹下解决方案 C++_Example.sln；

2. 在右侧解决方案资源管理器中，双击打开 C++_Example.sln 项目源文件目录下的 main.cpp；

3. 每个示例功能已封装在独立的函数中，使用时取消对应例程的注释并保存、选择编译架构再点击运行。

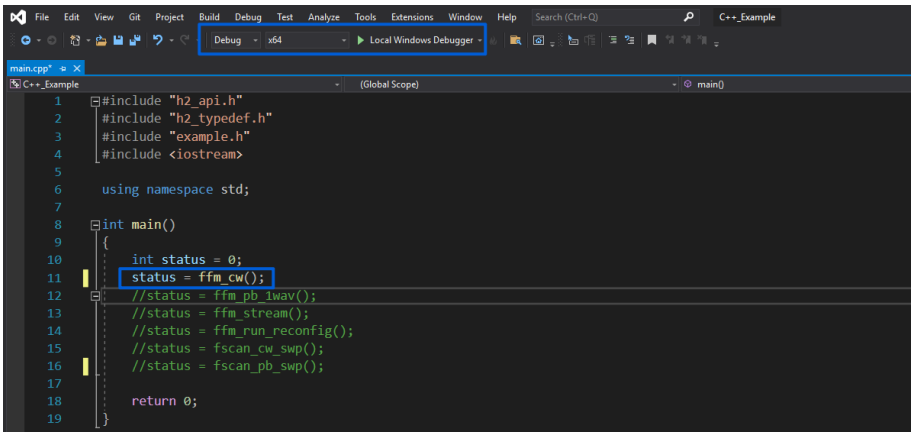


图 6 运行 ffm_cw 示例

1.2 Qt

1.2.1 配置开发环境

下文以配置 x64 架构的开发环境为例。

1. 创建一个新文件夹（下文以 QtTest 为例），用于存放项目文件（**注意路径勿包含中文**）；

2. 将随寄 U 盘中“Windows\api\x64”文件夹的内容拷贝至刚创建的“QtTest\api”文件夹中；

3. 打开 Qt Creator，点击“文件”->“新建文件或项目”；

4. 在项目中点击“Application(Qt)”，选择“Qt Widgets Application”并点击“Choose...”；

5. 填写项目名称（例如 test），点击“浏览”按钮，选择之前创建的 QtTest 文件夹，再点击“下一步”。

6. 选择“qmake”，继续点击“下一步”至“Kit Selection”界面。在此界面中选择一个构建环境，再点击“下一步”；

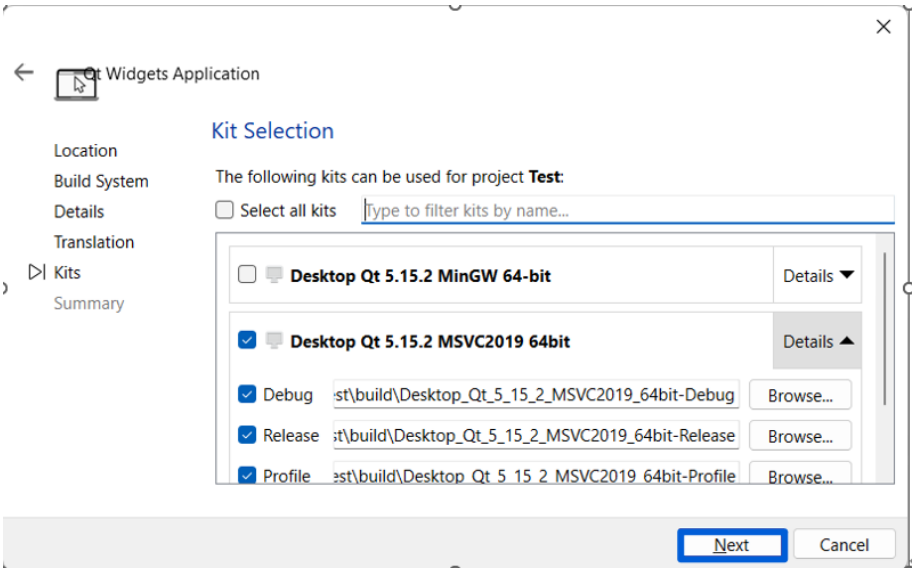


图 7 选择构建环境

7. 点击“完成”，创建项目；

8. 在 Qt Creator 主界面中右击“test”项目，选择“添加库...”->“外部库”->“下一步”；

9. 点击浏览库文件，选择“QtTest\api”中的 h2_api.lib 库，并点击“打开”；

10. 取消勾选 Windows 内所有选项，然后点击静态库，最后选择 Windows 平台，点击下一步；

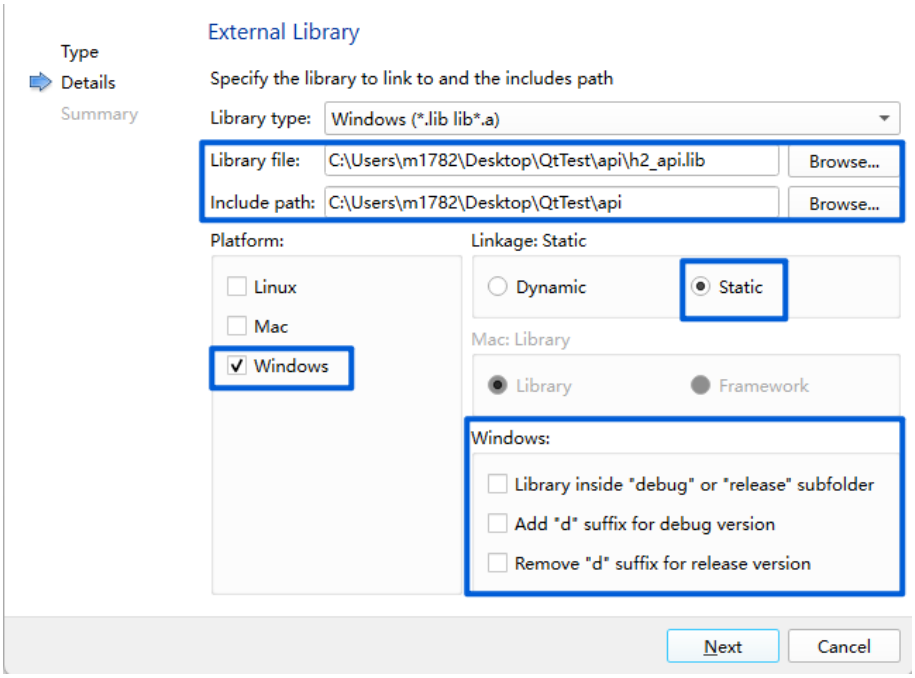


图 8 添加外部库

11. 点击“完成”，添加外部库，然后删除“test.pro”文件中最后一行“else:win32-g++: PRE_TARGETD EPS += $$PWD/../api/libh2_api.a”代码，保存后可正常编写项目代码。

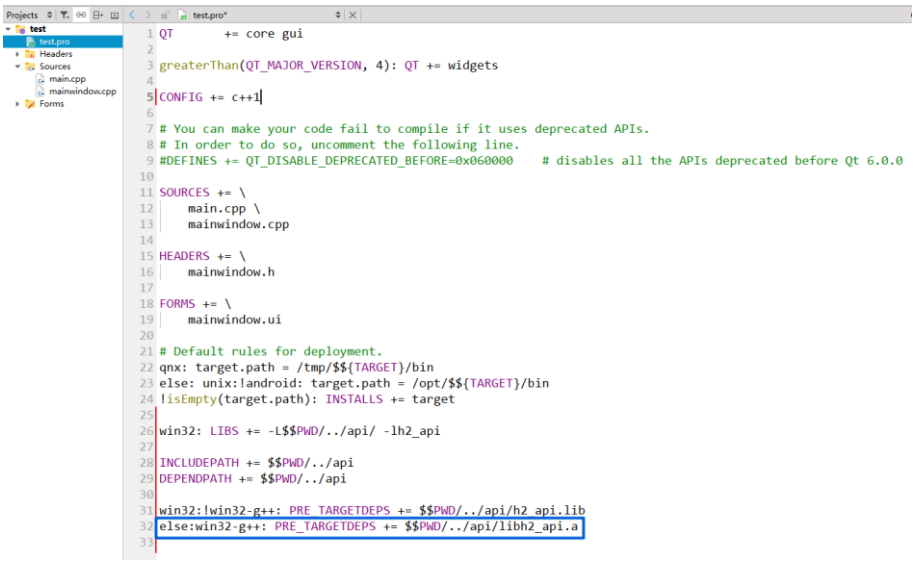


图 9 修改 test.pro 文件

12. 编写完代码后，点击运行。程序正确运行示意如下所示。

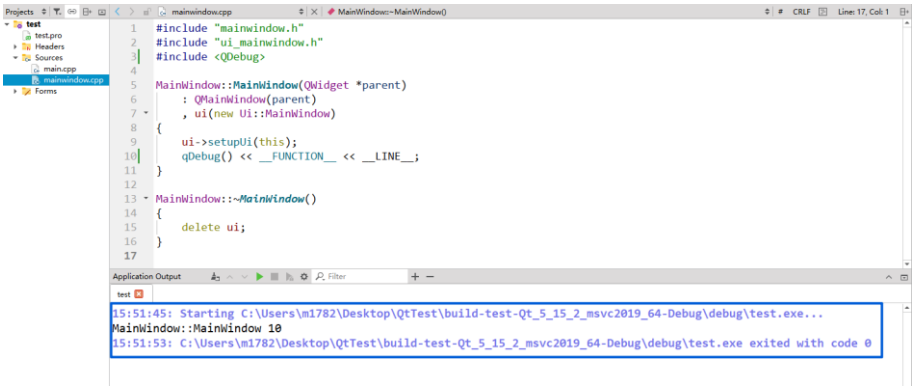


图 10 正确运行示意

1.2.2 范例使用流程

注：同一时间只能运行一个范例，范例与软件不可同时运行。

随寄 U 盘中所有 Qt 范例使用流程一致。下文以介绍 Qt_Example 项目的使用为例：

1. 使用 Qt Creator 打开随寄 U 盘“Windows\example\Qt\Qt_Example”文件夹下 Qt_Example.pro 文件（项目存放路径勿包含中文）；
2. 点击“项目”，为项目配置一个 64 位的构建环境（随寄范例中使用的是 64 位的库文件，若需使用 32 位的构建环境，请将“Qt\api”文件夹中的库替换为“\Windows\api\x86”中 32 位库）；

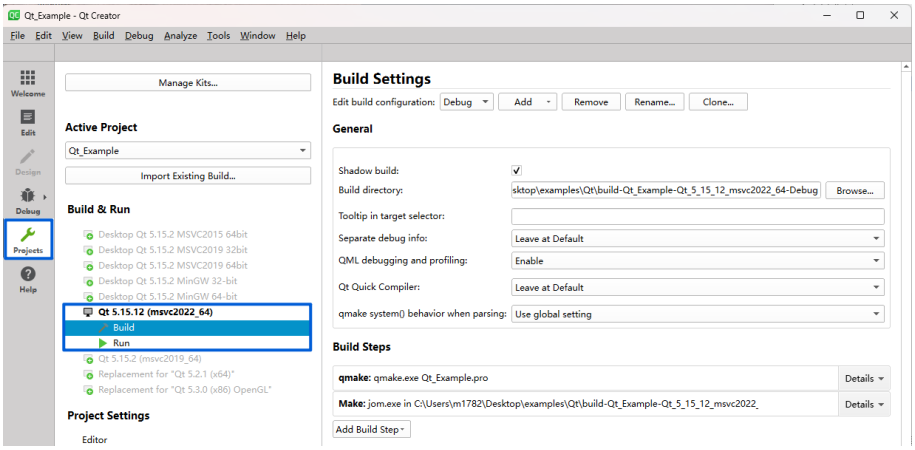


图 11 配置 64 位的构建环境

3. 环境配完后，点击“编辑”，打开“Qt_Example”项目中 Sources 文件夹下的 main.cpp，取消相关函数的注释并保存，点击运行，以运行不同的示例;

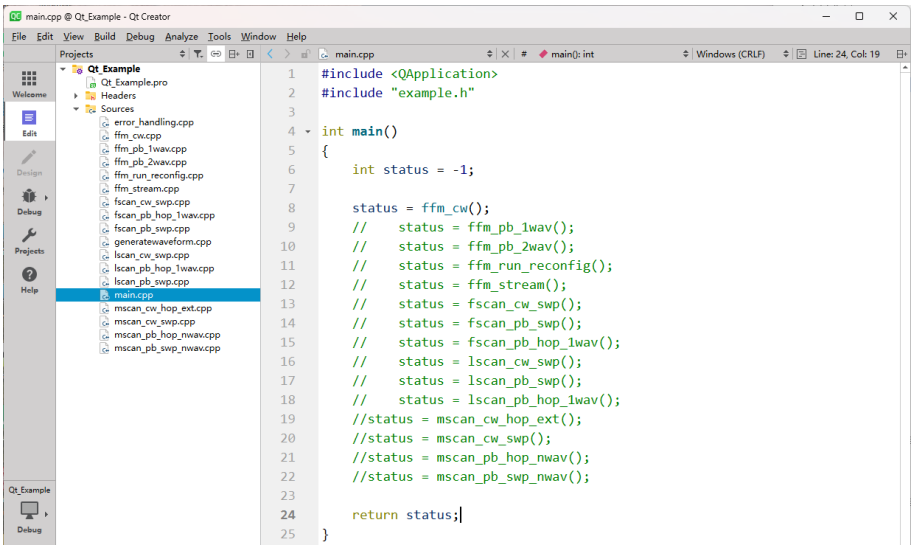


图 12 运行 Qt 示例

1.3 Python

1.3.1 配置开发环境

1. 在桌面创建一个新文件夹（例如 test）;

2. 打开随寄 U 盘，进入“\Windows\example\Python”，将“api”文件夹与“h2_api.py”文件复制到刚创建的 test 文件夹中;

3. 打开 Visual Studio Code，点击“文件”->“打开文件夹”，选择并打开刚创建的 test 文件夹。

4. 点击左侧资源管理器面板中的“新建文件”图标，创建新的 Python 文件（如 test.py），后续可正常编

写代码。

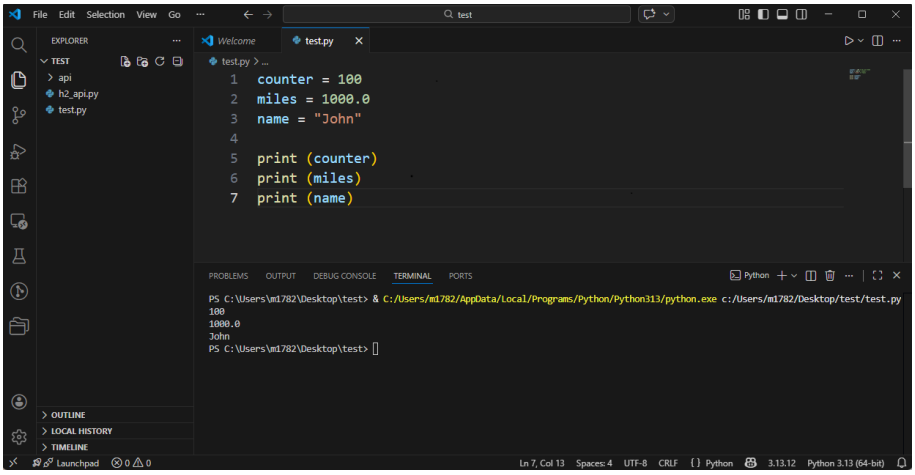


图 13 新建 Python 工程

1.3.2 范例使用流程

1. 使用 Visual Studio Code 或其他编译器打开随寄 U 盘中“Windows\example\Python”项目文件夹。其中 h2_api.py 为动态链接库在 Python 中的映射文件，其他文件为示例范例；

2. 正确配置 Python 环境后，选择任意示例程序（如 ffm_cw.py），直接运行该文件，程序运行成功示意如下所示：

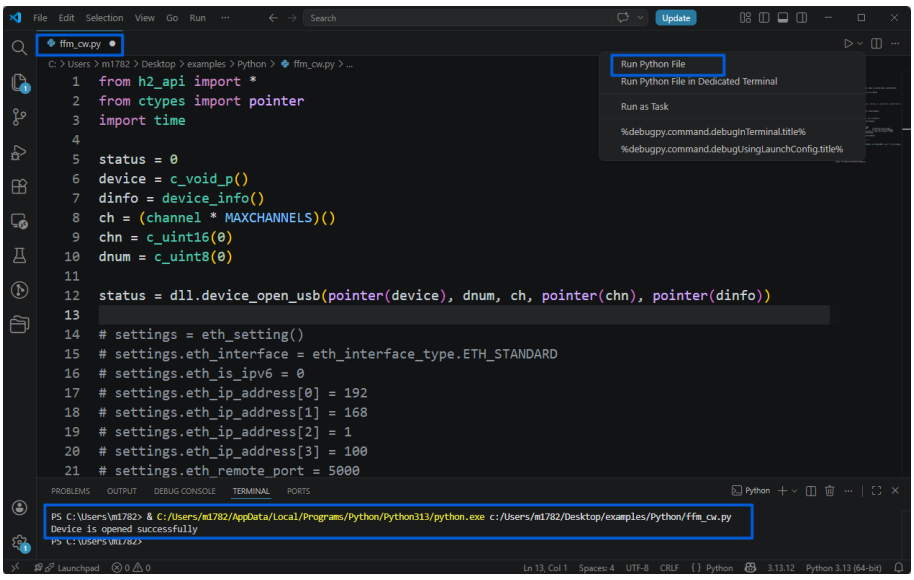


图 14 运行 ffm_cw.py

1.4 C#

1.4.1 配置开发环境

打开 Visual Studio Installer，勾选.NET 桌面开发组件与通用 Windows 平台开发组件并点击修改，以保证 Visual Studio 2019 具有 C#开发环境。

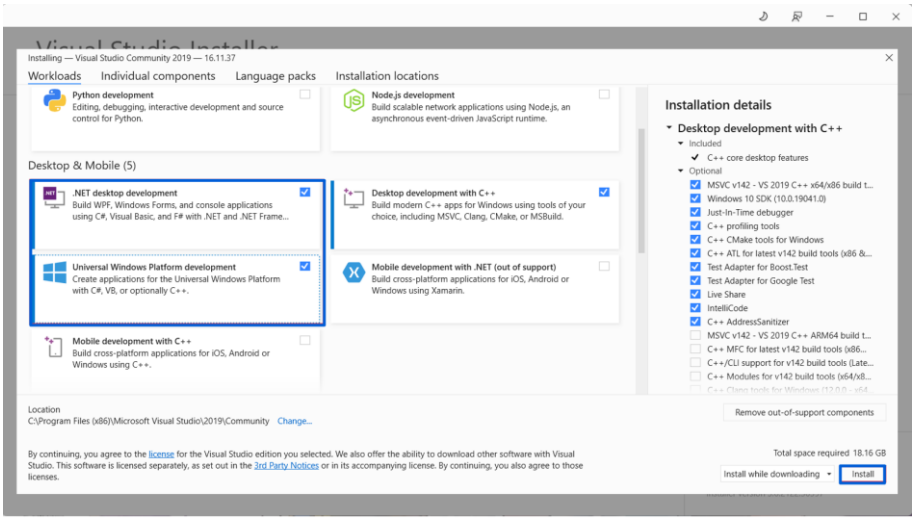


图 15 Visual Studio 中配置 C#开发环境

1. 打开 Visual Studio 2019，点击“创建新项目”；
2. 选择 C#的“控制台应用(.NET Framework)”，并点击“下一步”；
3. 填写项目名称（例如 ConsoleApp1）与位置，取消勾选“将解决方案和项目放在同一目录中”。框架选择“.NET Framework 4.5”，最后点击“创建”；
4. 创建完成后，右击解决方案“ConsoleApp1”，选择“添加”->“新建项目”；
5. 在“添加新项目”对话框中，项目类型选择“库”，再选择 C#的“类库(通用 Windows)”，并点击“下一步”；

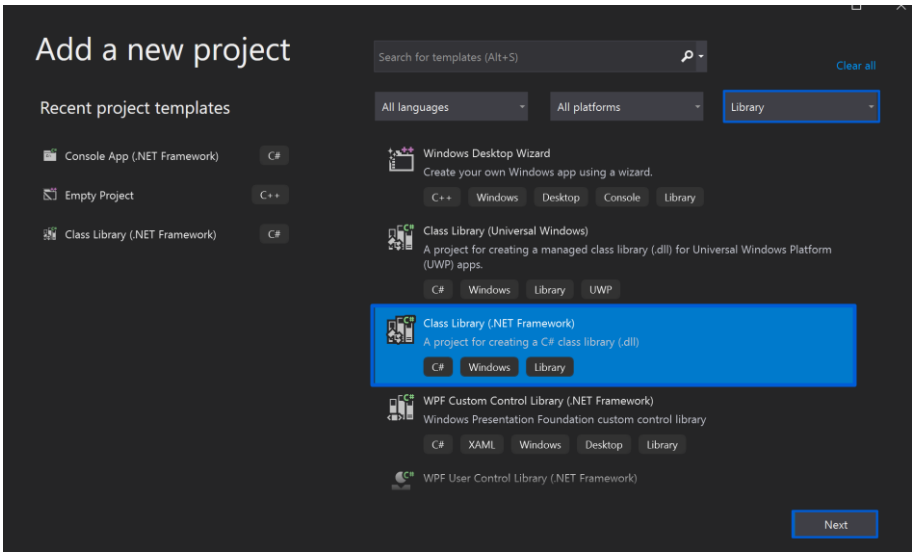


图 16 添加新项目

6. 按需求修改项目名称（如 api），位置保持默认，框架选择“.NET Framework 4.5”，点击“创建”；

7. 右击 ConsoleApp1，选择“属性”，在项目属性窗口中，点击“生成”选项卡，将“目标平台”修改为“x86”，然后点击“调试”选项卡，在“工作目录”中输入“api\”，点击弹窗中的“OK”确认，最后保存并关闭属性窗口；

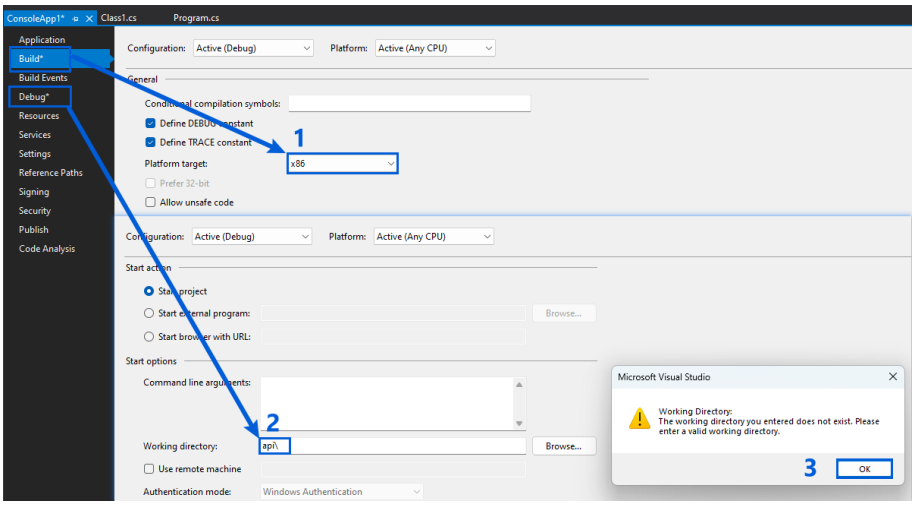


图 17 配置项目属性

8. 右击类库 api，选择“属性”，在库属性窗口中，点击“生成”，将“目标平台”修改为“x86”并保存；

9. 将随寄 U 盘中“\Windows\example\C#”文件夹下的 api.cs 文件内容复制到项目类库中的 Class1.cs 文件中，替换代码，并保存；

10. 选择 ConsoleApp1 项目，右击“引用”，选择“添加引用”，勾选项目中的“api”类库，点击“确定”，

确认添加类库引用；

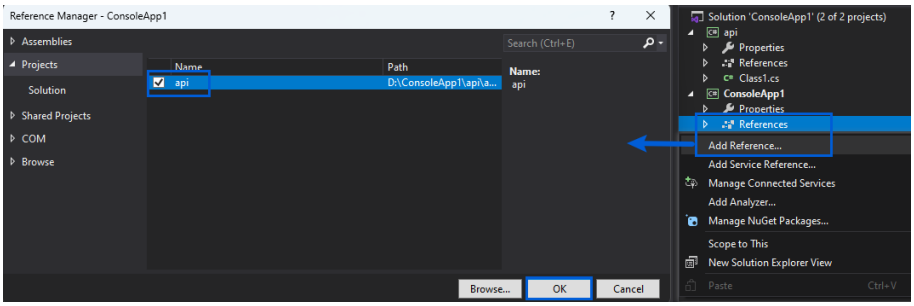


图 18 添加类库引用

11. 将随寄 U 盘中“\Windows\api\x86”下的内容复制到项目文件夹的“bin\Debug”目录下，并确保“api\CalFile”文件夹中包含所用仪器的校准文件；

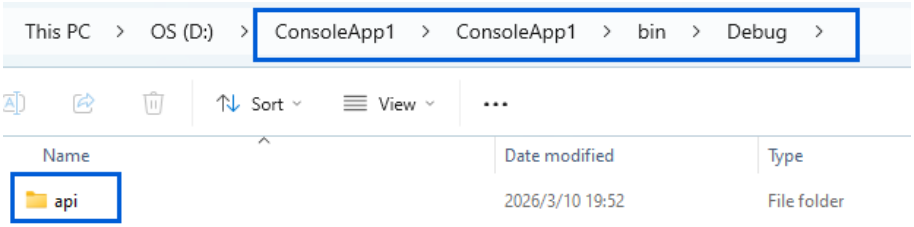


图 19 拷贝 api 文件

12. 可参照随寄 U 盘中 C#范例，根据需要编写代码。

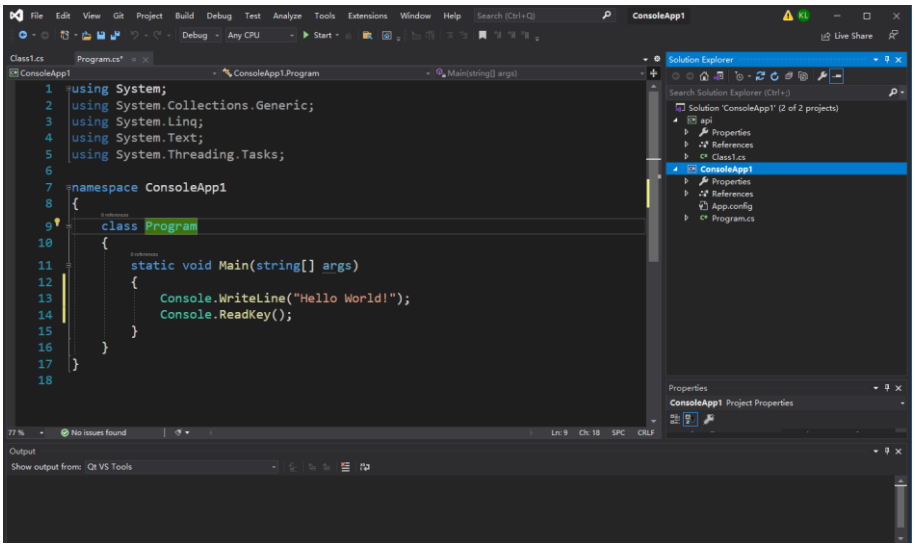


图 20 编写 C#代码

1.4.2 范例使用流程

1. 使用 Visual Studio 打开随寄 U 盘“Windows\example\C#”文件夹下解决方案 CSharp_Example.sln；

2. 点击右侧 CSharp_Example 项目，点击其中的 Programe.cs 文件；

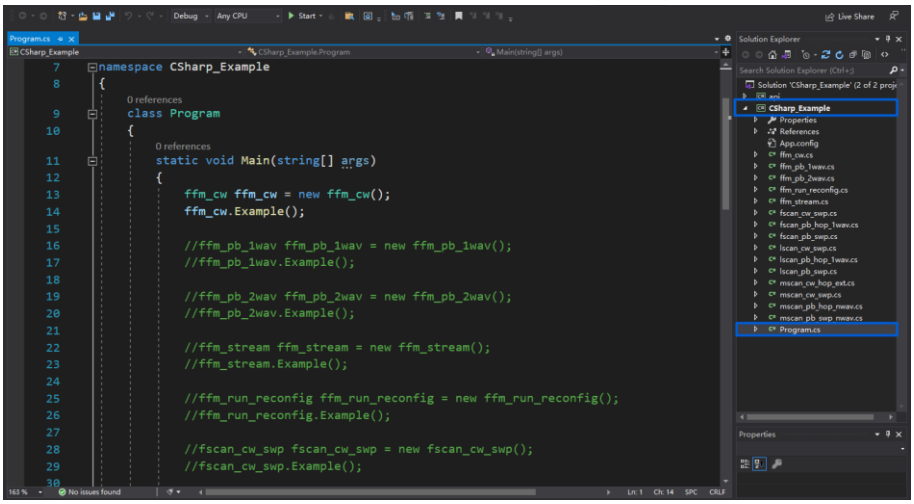


图 21 打开 Programe.cs 文件

3. C#随寄范例每个例程都封装在单独的类中，使用时取消注释即可（不可以同时使用多个范例）。

1.5 Matlab

下文将以 Matlab2016a 为例，介绍如何调用 64 位的 h2_api 动态链接库。

1.5.1 范例使用流程

1. 下载和安装 C++编译器；

2. 正确连接设备，打开 Matlab，将随寄 U 盘中“Windows\example\Matlab”文件夹的当前地址复制到 Matlab 的文件地址框中，并按下回车；

3. 点击左侧的 ffm_cw.m 文件，确认范例首行编译地址是否与本地系统上安装的编译器路径一致；

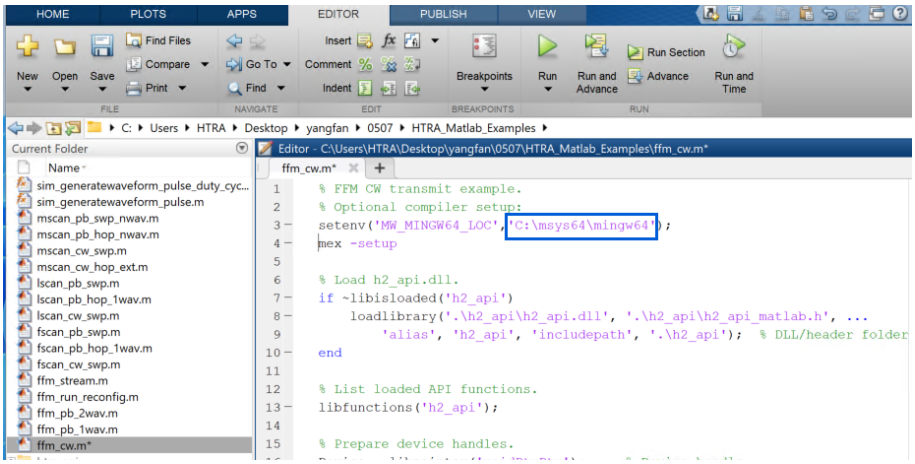


图 22 确认编译地址

4. 点击“Run”，程序正确运行示意如下，各范例的功能说明，请参考范例说明章节；

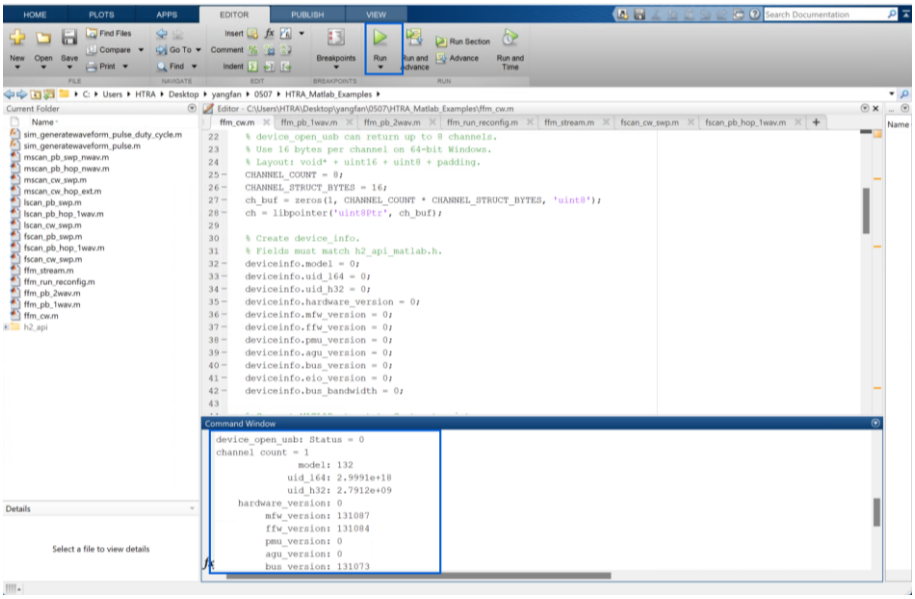


图 23 运行 ffm_cw.m 文件

注意：由于调用动态链接库必须使用 C++编译器，所以推荐使用 MSYS2 安装 g++编译器，可参考 https://www.msys2.org/进行下载与安装。

2. Linux 系统

2.1 系统运行环境检查

在 Linux 系统中使用设备时，首先需要按照下述流程确认当前 Linux 系统的系统架构、gcc 版本以及 GLIBC 版本是否已支持：

1. 打开终端，输入“uname -a”，查看 Linux 系统架构（下图中 Linux 系统架构为 x86_64）；
2. 输入“gcc -v”，查看系统 gcc 版本（下图中 gcc 版本为 7.5.0）；
3. 在终端输入“ldd --version”查看系统 GLIBC 版本（下图中 GLIBC 版本为 2.27）。

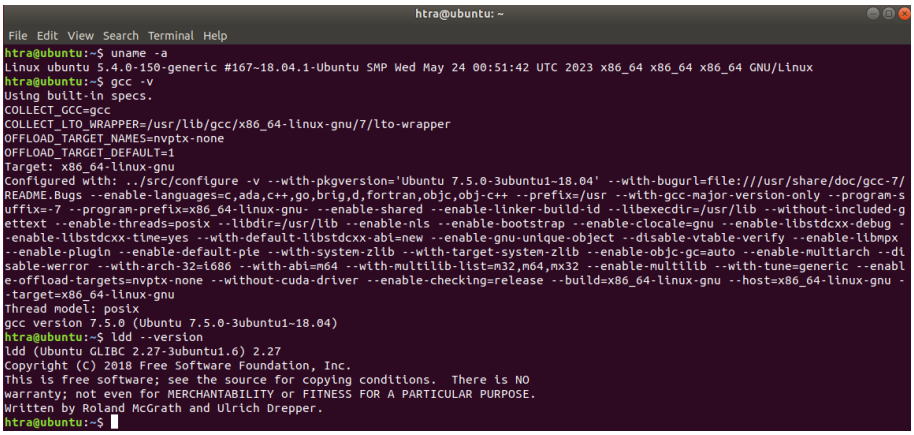


图 24 查看 Linux 系统的架构与版本

4. 根据终端信息对比表格确认当前环境是否已支持，若尚未支持，请联系技术支持人员。

表格 1 终端信息对比表

x86 处理器	支持 intel 与 AMD 处理器
ARM 处理器	aarch64（armv8）、armv7 处理器，例如：树莓派 4b、RK3399、RK3568、RK3588、T507、NIVIDA Jeston TX2
编译环境	gcc4.8、glib2.17 及以上
发行版	树莓派 4b 定制系统、ubuntu 18.04 等

2.2 随寄资料说明

目前随寄 U 盘 Linux 部分包含以下资料：

2.2.1 C++范例

“\Linux\C++”路径下 C++文件夹包含：

1. demo 文件夹：C++范例程序（使用方法见 C++范例使用章节）；
2. Makefile 文件：用于将范例程序编译为可执行文件的构建脚本；
3. bin 文件夹：用于存放参数限制文件以及范例程序构建生成的可执行文件。

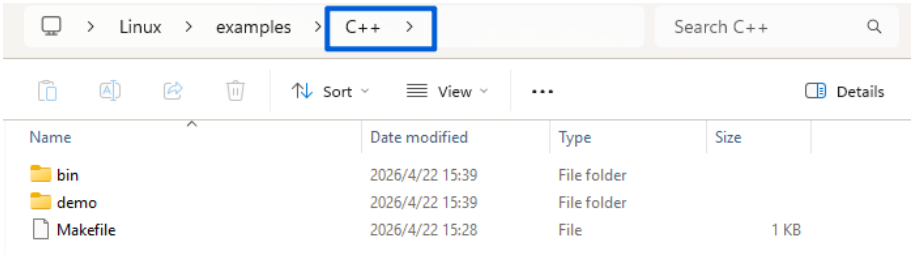


图 25 Linux 下 C++文件夹内容

2.2.2 Qt 范例

“\Linux\Qt”路径下 Qt 文件夹包含：

1. demo 文件夹：Qt 范例以及 pro 文件（使用方法见 Qt 范例使用章节）；
2. bin 文件夹：用于存放设备参数限制文件以及范例程序编译生成的可执行文件；
3. api 文件夹：用于存放动态链接库。

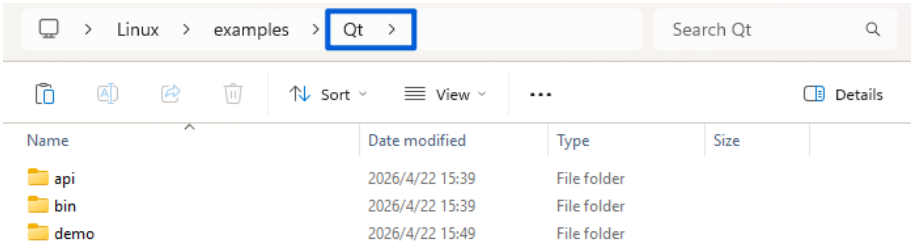


图 26 Linux 下 Qt 文件夹内容

2.2.3 Python 范例

“\Linux\Python”路径下 Python 文件夹具体包含：

1. Python 范例程序（使用方法见 Python 范例使用章节）。

2. CalFile 文件夹：存放设备参数限制文件。

3. api 文件夹：用于存放动态链接库。

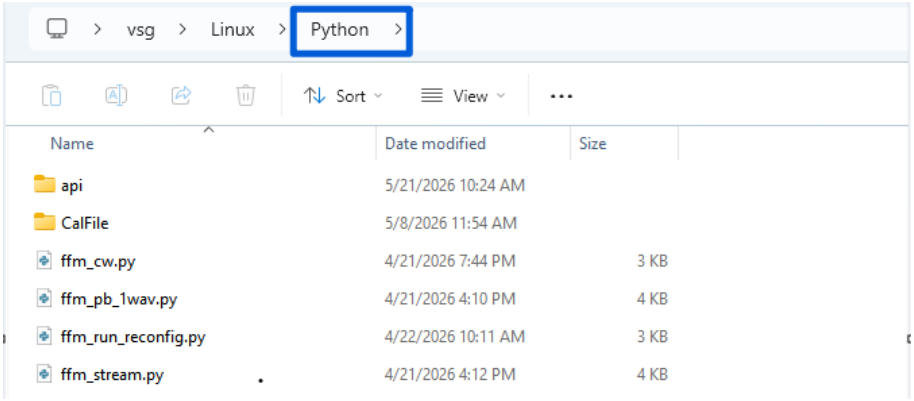


图 27 Linux 下 Python 文件夹内容

2.2.4 Install_H2_SDK

1. Install_H2_SDK 文件夹包含：

   install_h2_lib.sh：驱动配置脚本；

   h2api 文件夹：存放驱动、头文件和库。

2. Install_H2_SDK/h2api 文件夹中包含：

   configs 文件夹：驱动配置文件；

   inc 文件夹：头文件；

   lib 文件夹：动态链接库。

3. Install_H2_SDK/h2api/lib 文件夹中包含：

   aarch64 文件夹：aarch64 架构动态链接库；

   x86_64 文件夹：x86_64 架构动态链接库；

   armv7 文件夹：armv7 架构动态链接库。

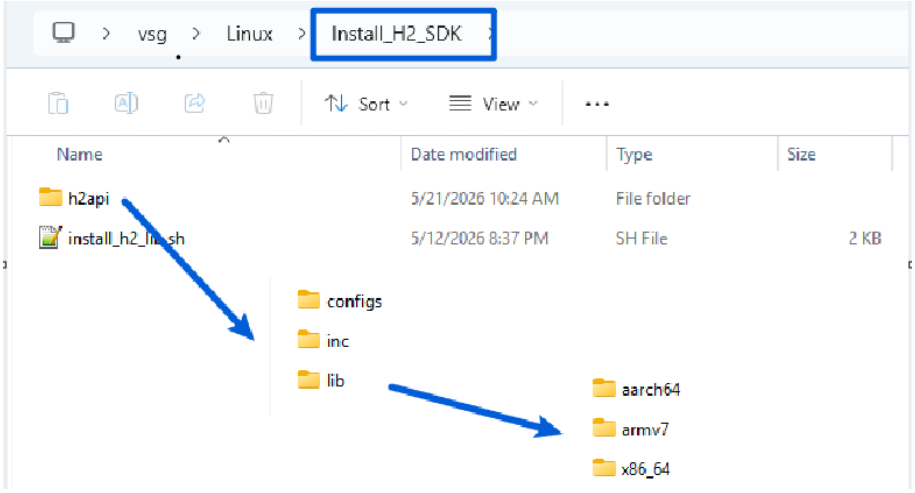


图 28 Linux 下 Install_H2_SDK 文件夹内容

2.3 配置驱动文件

在 Linux 中使用设备必须先进行驱动文件配置，具体流程如下：

1. 将 Install_H2_SDK 文件夹拷贝至 Linux 上位机，在 Install_H2_SDK 目录下打开终端，执行“sudo sh install_h2_lib.sh”配置驱动文件；

   注意：若特殊开发板无 sudo 命令，输入“sh install_h2_lib.sh”即可。

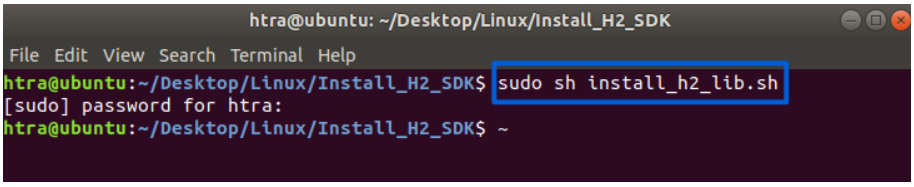


图 29 Linux 下配置驱动文件

2. 正确连接设备与 Linux 上位机，并确保设备正常供电（若上位机运行在虚拟机中，需先将设备连接至虚拟机，同时将 USB 兼容性设置为 USB3.1），在终端输入“lsusb”查看 USB 设备列表，若出现“ID: 6430（或 ID：04b5 或 ID：367f）”即为成功接入设备。

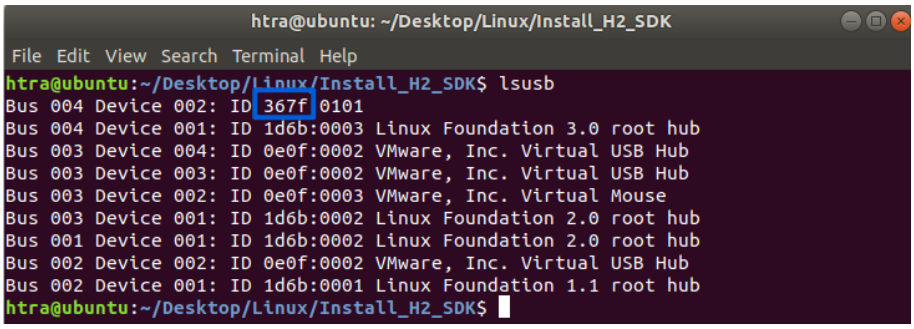


图 30 查看 Linux 下设备连接情况

2.4 C/C++

2.4.1 范例使用流程

使用前提：确保设备已正确连接，并按照配置驱动文件章节正确配置驱动文件。

可参考以下流程，使用随寄 U 盘中 C++范例（各示例具体功能说明参见范例说明章节）：

1. 选择需要编译的程序：将随寄 U 盘“Linux\C++”文件夹拷贝至上位机。双击打开 demo 文件夹下 main.cpp，将需要测试使用的范例注释取消；

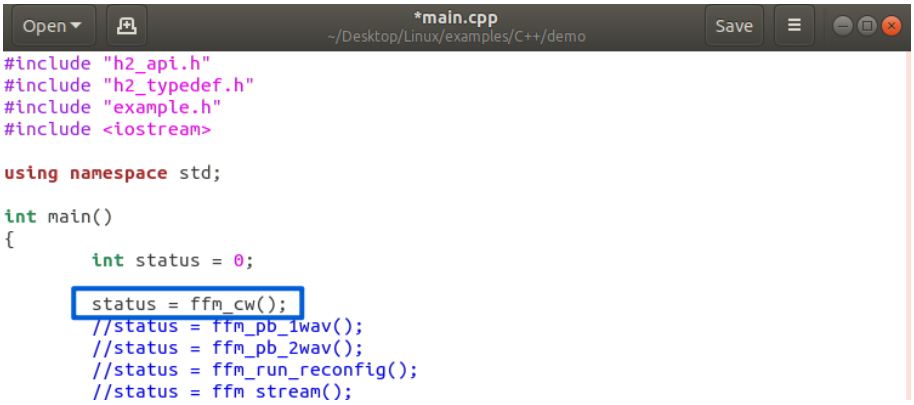


图 31 取消 ffm_cw 范例注释

2. 编译所选示例：按系统运行环境章节查看系统架构，在 C++文件夹下打开终端，按照上位机系统架构输入相应命令（下文以编译 ffm_cw 例程为例）；

   x86_64 架构输入：make Example=ffm_cw

   aarch64 架构输入：make TARG=aarch64 Example=ffm_cw

   armv7 架构输入：make TARG=armv7 Example=ffm_cw

3. 运行程序：在程序编译成功后打开终端输入：“./bin/ffm_cw”，运行所选范例。

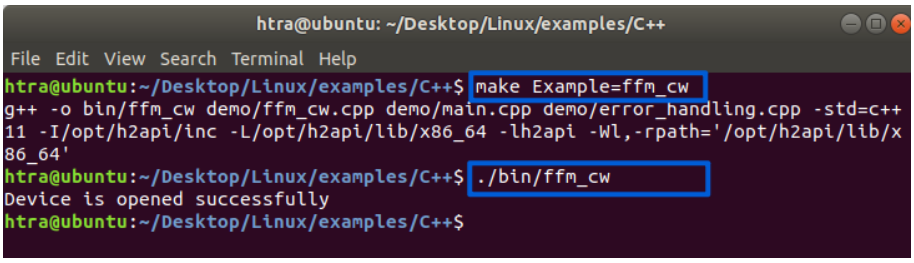


图 32 x86_64 架构下编译运行 ffm_cw 示例

2.4.2 创建与编译新项目

使用前提：保证设备正常接入且已按配置驱动文件章节正确配置驱动文件。

1．**编写代码**：

随寄 U 盘中提供的 Linux 动态链接库与 Windows 中完全一致，代码编写逻捷符合 API 编程指南即可。

2．**编译运行**：

(1) 创建一个新文件夹用于存放整个项目（如 C++_Test），然后在文件夹内创建 CalFile 文件夹用于存放参数限制文件，创建 h2 api 文件夹用于存放头文件以及动态链接库。

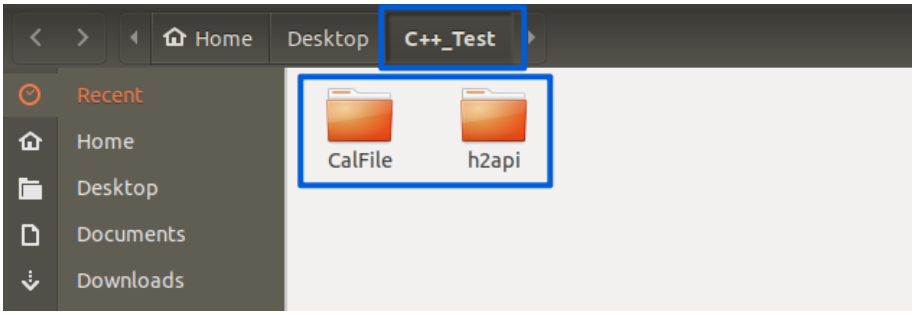


图 33 创建项目文件夹

(2) 在 h2api 文件夹下创建 inc 文件夹用于存放头文件，lib 文件夹用于存放动态链接库。

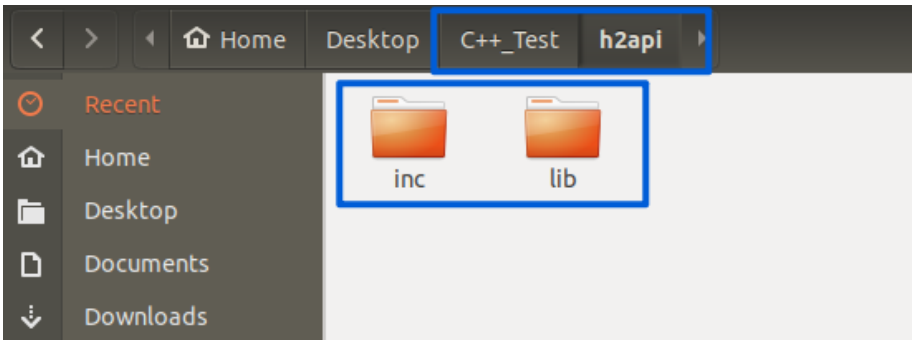


图 34 h2api 文件夹下创建 inc 与 lib 文件夹

(3) 将随寄 U 盘 CalFile 文件夹中的文件拷贝至刚创建的“C++_Test\CalFile”文件夹中。

(4) 将随寄 U 盘“Linux\Install_H2_SDK\h2api\inc”中的头文件拷贝至刚创建的“C++_Test\h2api\inc”文件夹中。

(5) 参照系统运行环境，查看系统架构。并依照 lib 文件夹介绍，拷贝其中对应架构文件夹下的文件至“C++_Test\h2api\lib”文件夹中。

(6) 在 lib 文件夹下打开终端，输入以下命令，对复制过来的动态链接库进行软链接（三种架构的动态链接库软链接指令无区别）：

下文 libh2api.so 库以 2.0.17 版本为例，其他版本修改版本号即可

```
ln -sf libh2api.so.2.0.17 libh2api.so.2
ln -sf libh2api.so.2 libh2api.so
ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0
ln -sf libusb-1.0.so.0 libusb-1.0.so
```

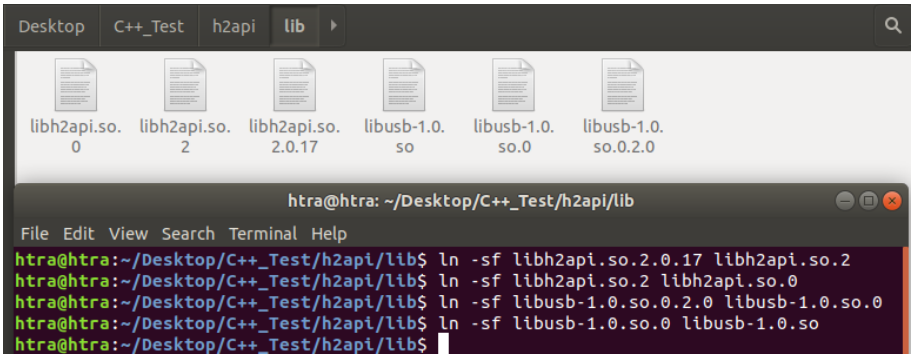


图 35 软链接动态链接库

(7) 将编写好的代码文件存放于 C++_Test 最外层文件夹中；

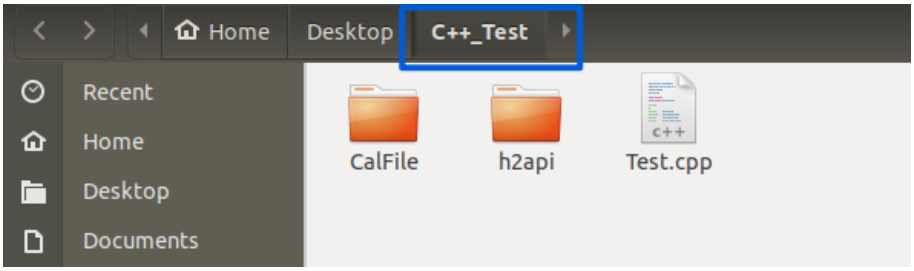


图 36 放置代码文件

(8) 编译生成可执行文件：根据系统运行环境检查中确认的系统架构，在 C++_Test 文件夹下打开终端，输入对应的编译命令。

下文以 Test.cpp 为例，在不同的操作系统上编译 Test.cpp，并生成名为 Test 的可执行文件：

x86_64 架构输入：

g++ -o Test Test.cpp -std=c++11 -I ./h2api/inc -L ./h2api/lib -lh2api -Wl,-rpath='./h2api/lib'

aarch64 架构输入：

aarch64-linux-gnu-g++ -o Test Test.cpp -std=c++11 -I ./h2api/inc -L ./h2api/lib -lh2api -Wl,-rpath='./h2api/lib'

armv7 架构输入：

arm-linux-gnueabihf-g++ -o Test Test.cpp -std=c++11 -I ./h2api/inc -L ./h2api/lib -lh2api -Wl,-rpath='./h2api/lib'

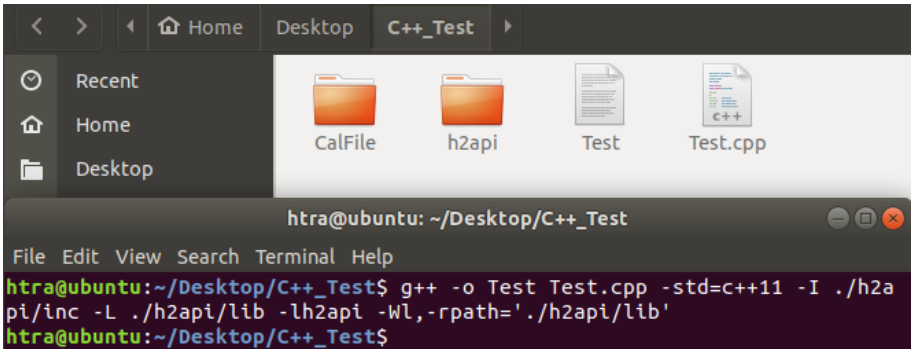


图 37 编译生成可执行文件

(9) 运行程序：输入./Test 启动可执行文件。

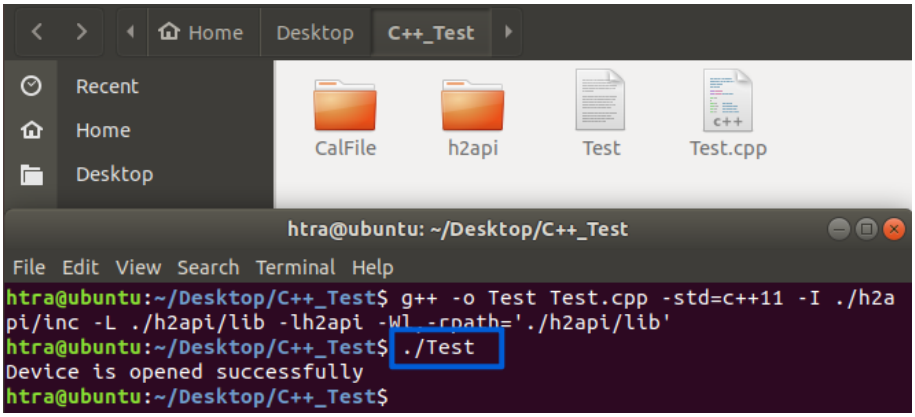


图 38 运行 Test 可执行文件

2.4.3 交叉编译

在上位机有交叉编译链的前提下，若想要交叉编译使用设备，请参考以下流程。

下文以在 x86_64 的上位机交叉编译 aarch64 可执行程序为例：

1. 生成目标架构可执行文件：

（1） 按照创建与编译新项目章节中步骤(1)-(8)，创建项目，放置头文件和交叉编译目标架构(aarch64)的库文件。随后进行软链接与程序编写存放，并使用目标架构编译命令编译可执行文件。如下图所示预计编译 aarch64 架构的可执行文件时使用 aarch64 系统的编译命令；

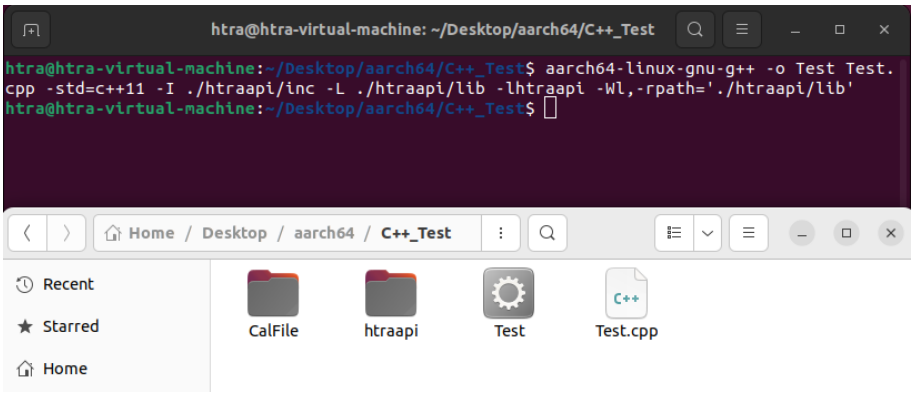


图 39 交叉编译 aarch64 架构的可执行程序

（2） 生成可执行文件后，输入“file Test”查看可执行程序的架构，可以看到当前可执行程序架构为 aarch64；

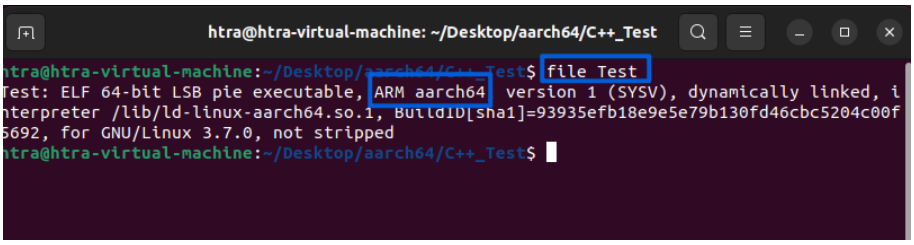


图 40 查看可执行程序架构

2. 在 aarch64 架构上位机中运行可执行程序：

（1） 进入项目所在目录，执行“zip -r C++_Test.zip C++_Test”将整个 C++_Test 文件夹打包为 zip 压缩包，然后将生成的压缩包拷贝至 aarch64 架构的上位机中；

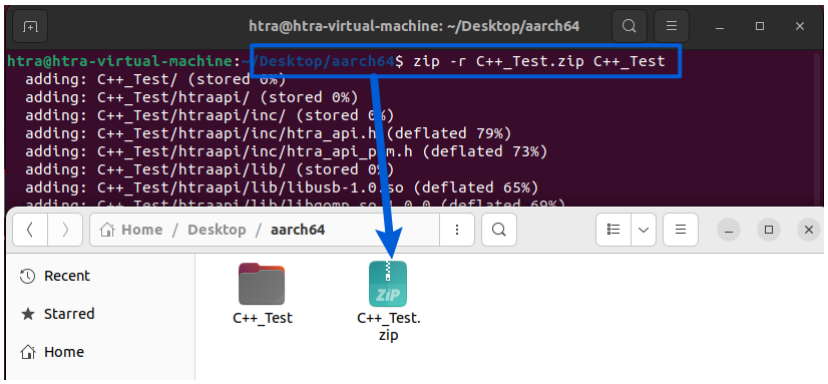


图 41 压缩所创建的项目

（2） 在 aarch64 上输入“unzip C++_Test.zip”解压项目。

（3） 参照配置驱动文件章节，在 aarch64 上位机中配置驱动文件。

（4） 配置好驱动文件后，进入 C++_Test 文件夹，终端输入./Test 运行程序。

2.5 Qt

2.5.1 范例使用流程

使用前提：保证设备正常接入且已按配置驱动文件章节正确配置驱动文件。

下文以在 ubuntu18.04 系统，x86_64 架构下运行 Qt 范例为例。

1. 将随寄 U 盘中“Linux\Qt”文件夹拷贝至上位机；

2. 参照系统运行环境检查查看系统架构，并将随寄 U 盘“Linux\Install_H2_SDK\h2api\lib”文件夹中，对应架构文件夹中的内容复制至上位机“Qt\api”文件夹中；

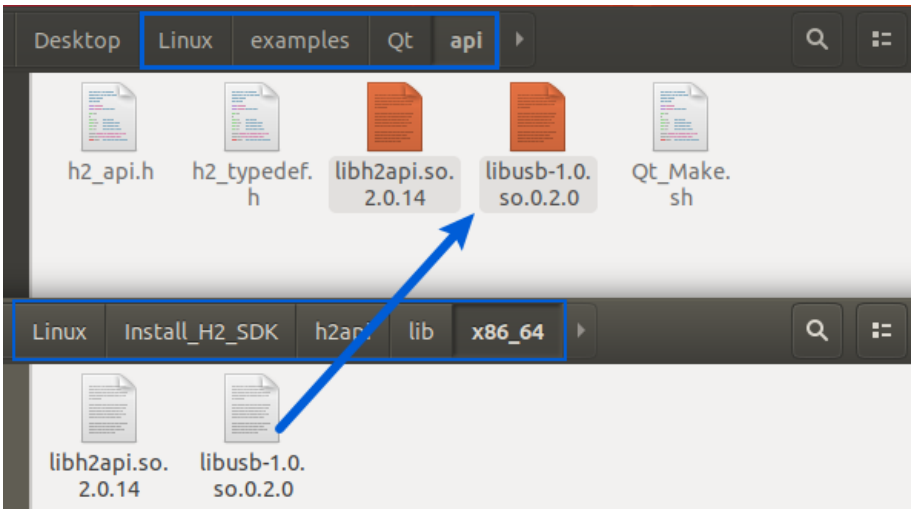


图 42 拷贝对应架构的动态链接库

3. 在 api 文件夹下打开终端，输入“sudo sh Qt_Make.sh”，按提示输入密码授权创建软链接；

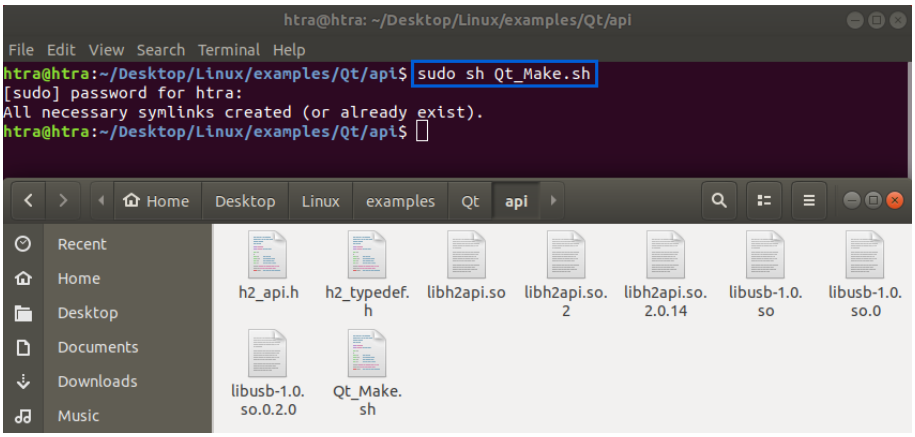


图 43 创建动态链接库的软链接

4. 使用 QT Creator 打开“Qt\demo”中 demo.pro 文件，并配置项目的构建环境；

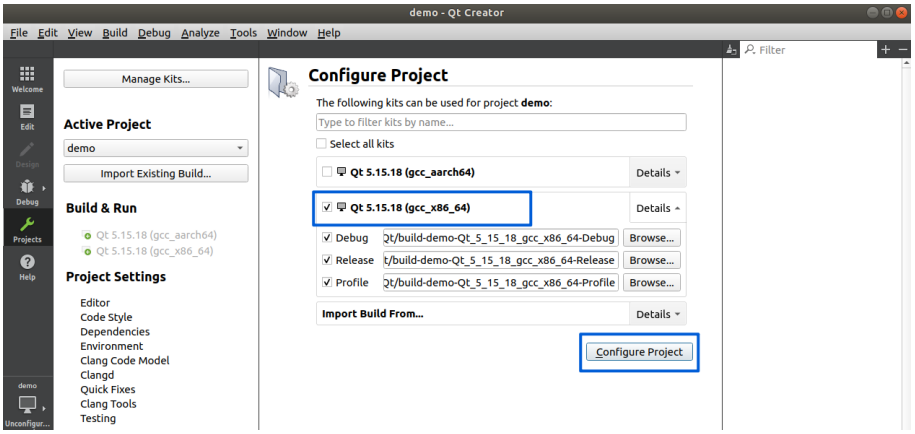


图 44 构建环境

5. 点击“编辑”，打开“demo”项目中 Sources 文件夹下的 main.cpp，取消相关函数的注释并保存，点击运行，以运行不同的示例。

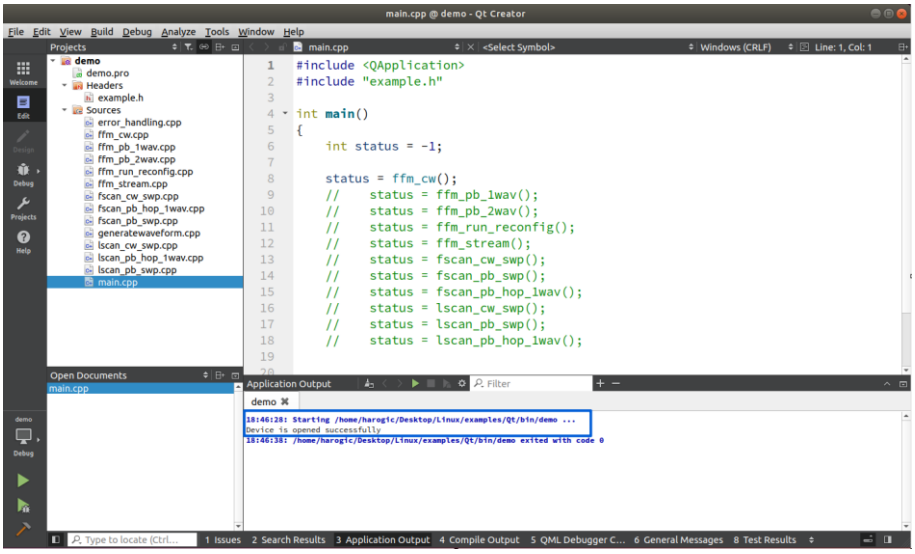


图 45 运行 ffm_cw 示例

2.5.2 创建与编译新项目

在已按配置驱动文件章节，正确配置驱动文件的前提下，若想要创建 Qt 项目编译使用，请参考以下流程：

代码部分遵循 API 编程指南，窗体程序创建流程如下：

1. 创建一个新文件夹用于存放整个项目（如 QtTest），内部包括 bin（存放校准与生成的可执行文件）与 h2api（存放头文件和动态链接库）文件夹；
2. 将仪器对应随寄 U 盘中“CalFile”文件夹拷贝至“QtTest\bin”文件夹中；
3. 将随寄 U 盘中“Linux\Install_H2_SDK\h2api\inc”文件夹中的头文件拷贝至“QtTest\h2api”文件夹中；
4. 将随寄 U 盘中“Linux\Install_H2_SDK\h2api\lib”文件夹下对应系统架构文件夹中的动态链接库拷贝至“QtTest\h2api”文件夹中（此处以 x86_64 架构的上位机为例）；

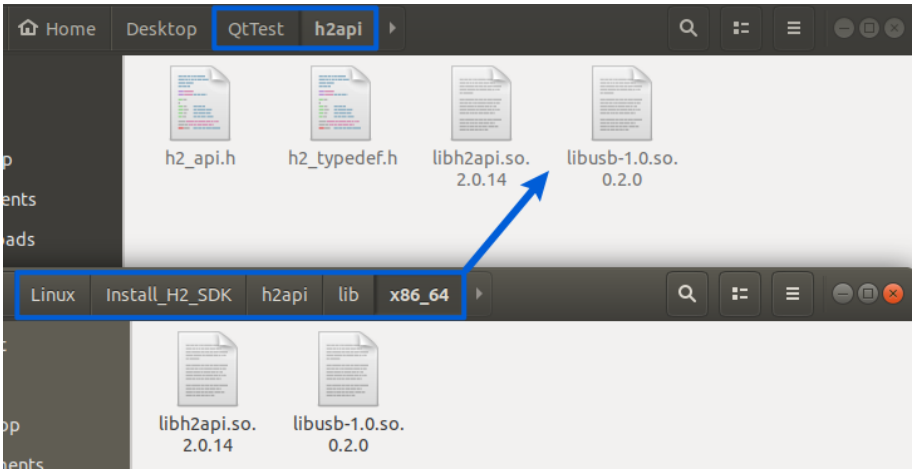


图 46 拷贝对应架构的动态链接库

5. 在 h2api 文件夹下打开终端，依次输入以下命令，对复制过来的动态链接库进行软链接（不同架构的动态链接库软链接指令无区别）：

下文 libh2api.so 库以 2.0.14 版本为例，其他版本修改版本号即可

```
ln -sf libh2api.so.2.0.14 libh2api.so.2
ln -sf libh2api.so.2 libh2api.so
ln -sf libusb-1.0.so.0.2.0 libusb-1.0.so.0
ln -sf libusb-1.0.so.0 libusb-1.0.so
```

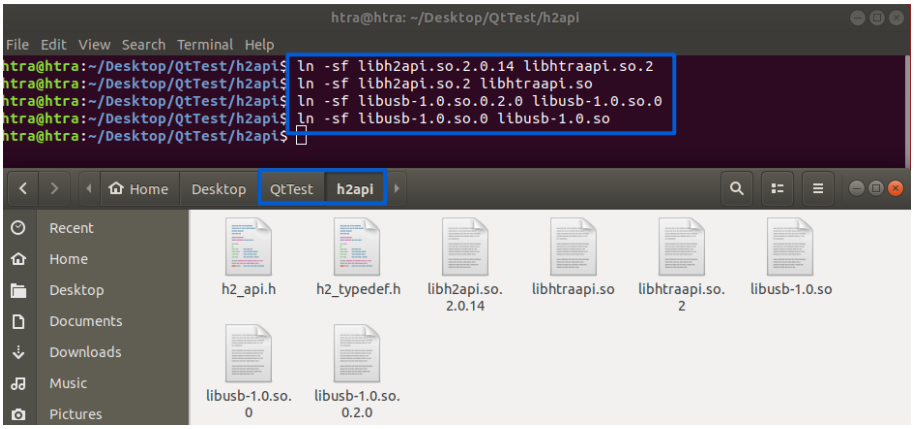


图 47 h2api 中对动态链接库进行软链接

6. 打开 Qt Creator，点击“文件”->“新建文件或项目”；

7. 在项目中点击“Application”，选择“QT Widgets Application”并点击“Choose…”；

8. 填写项目名称（例如 test），点击“浏览”按钮，选择之前创建的 QtTest 文件夹，再点击“下一步”；

9. 选择“qmake”，继续点击“下一步”至“Kit Selection”界面。选择一个构建环境，点击“下一步”；

10. 点击“完成”，创建项目；

11. 在 QT Creator 主界面，右击“test”项目，选择“添加库…”->“外部库”->“下一步”；

12. 点击浏览库文件，选择“QtTest\h2api”中的 libh2api.so，并点击“打开”，选择“Linux”平台，点击“下一步”。

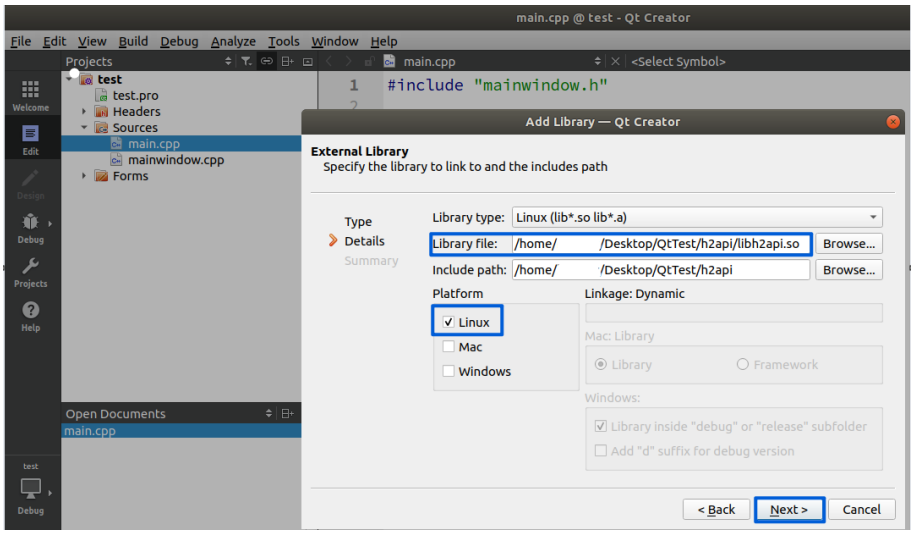


图 48 选择 Linux 平台

13. 点击“完成”以添加外部库；

14. 在 Test.pro 中，CONFIG += c++11 之后添加“DESTDIR = $$clean_path($$PWD/../bin)”，指定可执行程序生成位置；

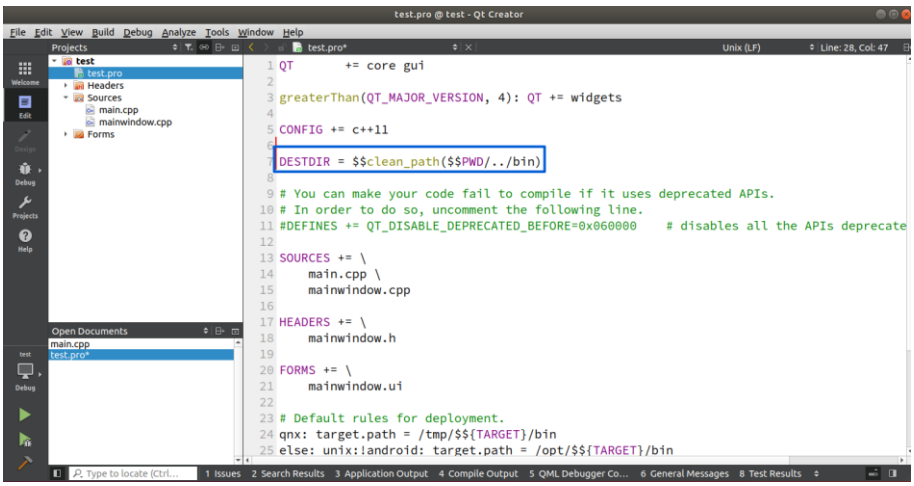


图 49 添加可执行程序生成位置

15. 在库文件之后，添加“ -Wl,-rpath,$$PWD/../h2api”，指定可执行程序的链接库路径；

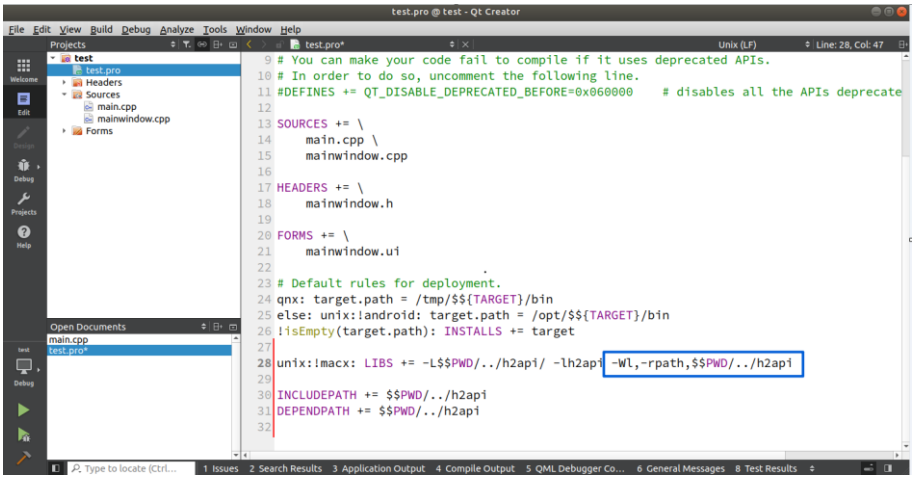


图 50 指定可执行程序的链接库路径

16. 保存 test.pro 文件，之后在 mainwindow.cpp 中编写代码并点击运行，正常运行界面如下所示；

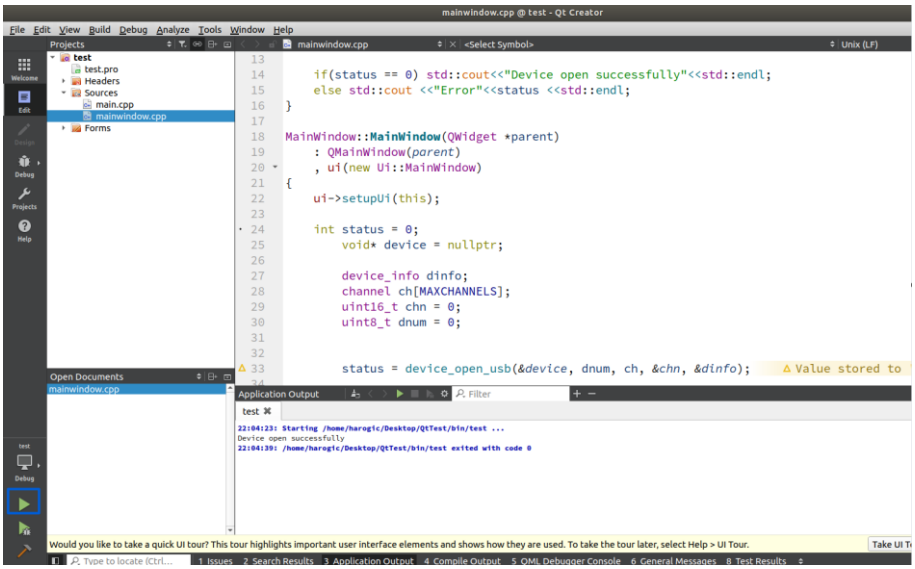


图 51 运行代码

17. 关闭 Qtcreater，进入 QtTest\bin 文件夹，打开终端，输入./test 即可运行可执行程序。

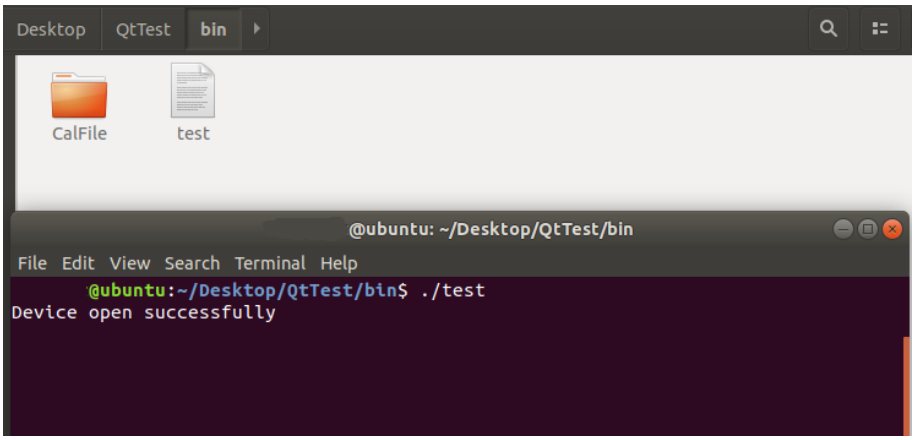


图 52 运行可执行程序

2.5.3 交叉编译

在上位机有交叉编译链的前提下，若想要交叉编译使用设备，请参考以下流程（此处以在 x86_64 的上位机交叉编译 aarch64 可执行程序为例）：

1. 生成目标架构可执行文件：

（1） 按照 QT 创建与编译新项目中窗体程序创建流程的 1-9 步，创建项目并放置参数限制文件与头文件、交叉编译目标架构（aarch64）的库文件、对动态链接库进行软连接、在 QTCreator 创建程序，并选择 aarch64 的构建套件；

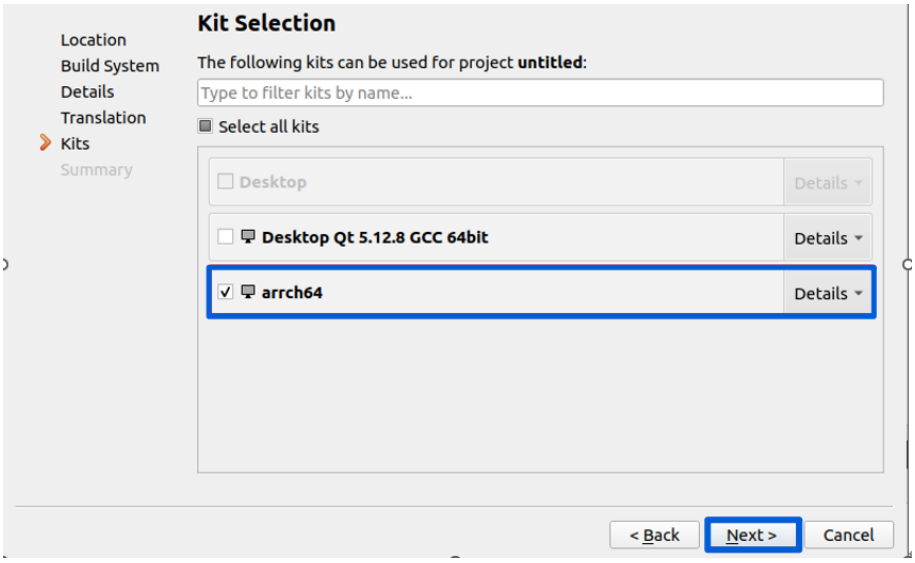


图 53 选择交叉编译目标架构的目标套件

(2) 按窗体程序创建流程第 10-16 步进行项目创建、库引用、可执行程序生成位置修改、项目编写及运行;

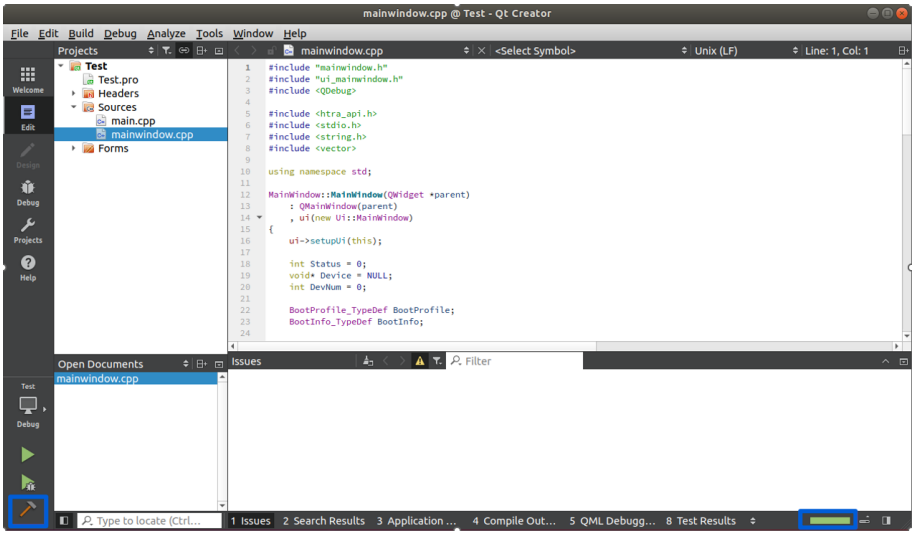


图 54 构建 aarch64 架构的可执行程序

(3) 构建好可执行程序后，在 QtTest\bin 文件夹下打开终端输入 file Test 查看可执行程序的架构（此处可执行程序名称为 Test，因此输入 file Test）。

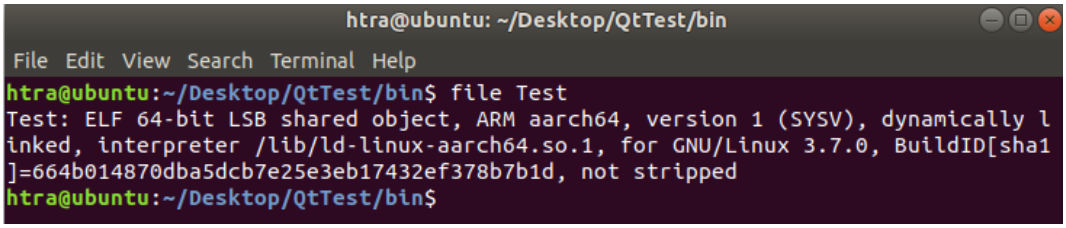


图 55 查看可执行程序的架构

2. 在 aarch64 架构上位机中运行可执行程序：

（1） 进入项目所在目录（此处示例项目位于桌面因此输入 cd Desktop/），执行“zip -r QtTest.zip QtTest”将整个 QtTest 文件夹打包为 zip 压缩包，然后将生成的压缩包拷贝至 aarch64 上位机中；

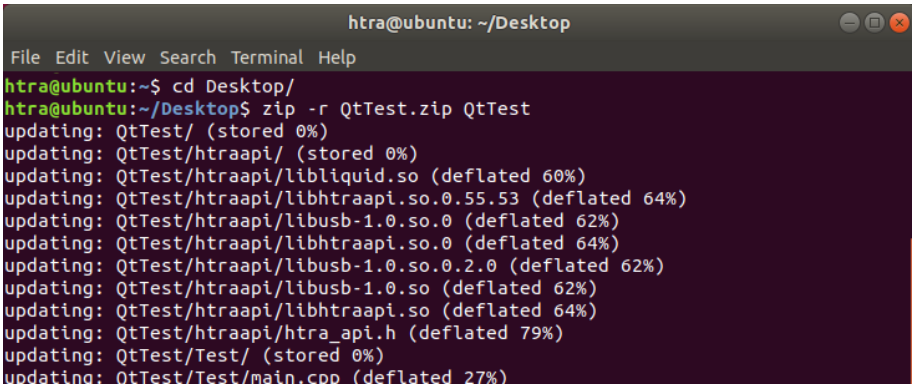


图 56 压缩所创建的项目

（2） 在 aarch64 上输入“unzip QtTest.zip”解压项目；

（3） 参照配置驱动文件章节，在 aarch64 上位机中配置驱动文件；

（4） 配置好驱动文件后，进入 QtTest 文件夹，终端输入“chmod 777 Test”为可执行程序提供权限，后续输入“./Test”即可运行程序。

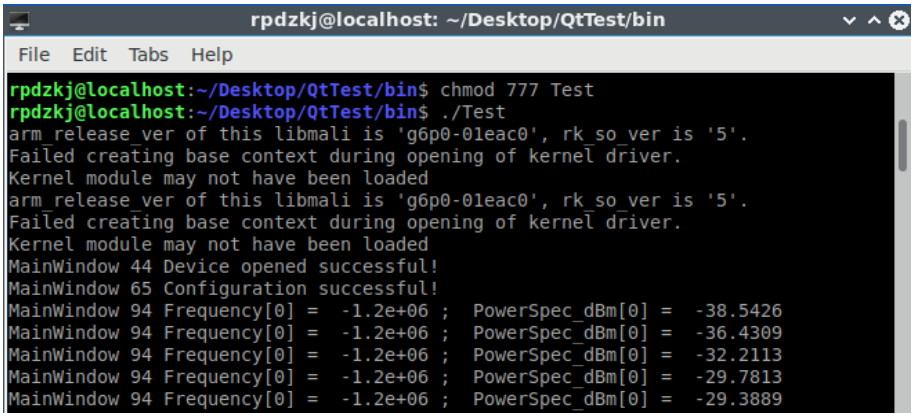


图 57 提供权限并运行可执行程序

2.6 Python

2.6.1 范例使用流程

使用前提：保证设备正常接入且已按配置驱动文件章节正确配置驱动文件。

1. 将随寄 U 盘中“Linux\Python”文件夹拷贝至上位机，在该文件夹下打开终端，

   输入“which python3”查看 Python 解释器位置（例如此处位置为/usr/bin）；

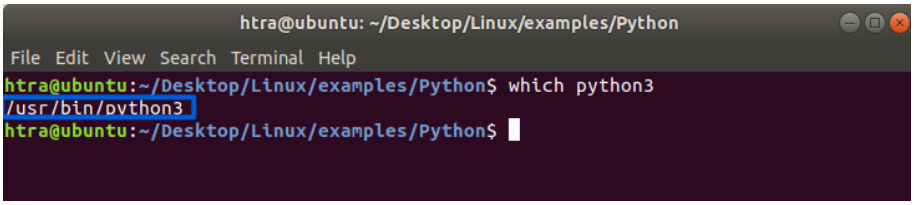


图 58 查看 Python3 解释器位置

2. 根据获取地解释器地址，输入“sudo cp -r CalFile /usr/bin”，将 CalFile 文件夹拷贝至 Python

   解释器同级目录（请跟据实际位置，修改上述拷贝路径）；

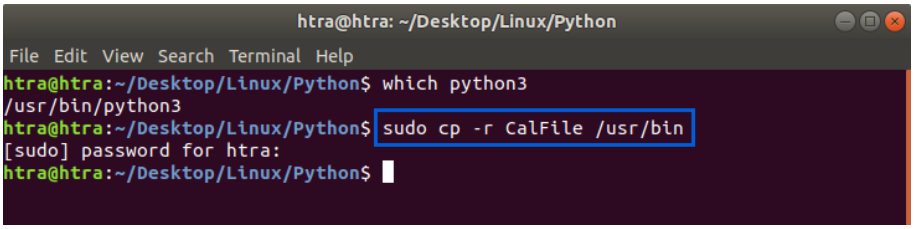


图 59 拷贝 CalFile 文件夹至解释器同级目录

3. 参照系统运行环境检查章节查看系统架构，并将随寄 U 盘“Linux\Install_H2_SDK\h2api\lib”文件

   夹中，对应架构文件夹中的内容复制至上位机“Python\api”文件夹中；

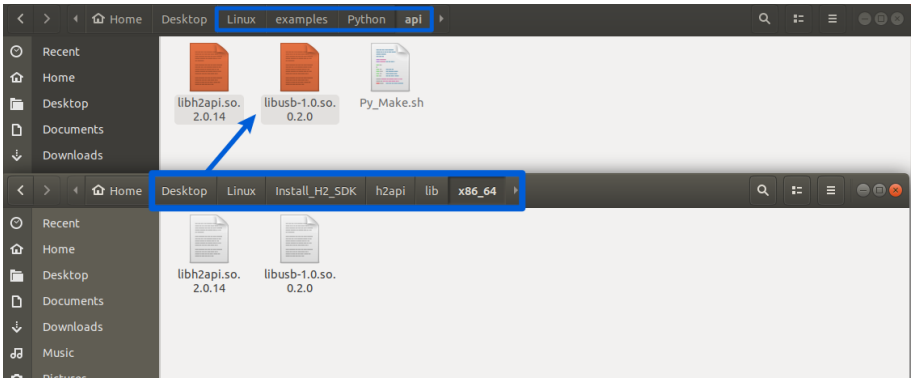


图 60 拷贝动态链接库

4. 在 api 文件夹下打开终端，输入“sudo sh Py_Make.sh”，根据提示输入密码，提供权限对库进行软

   链接。

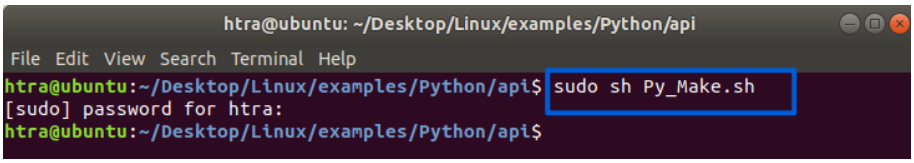


图 61 对库进行软链接

5. 在 Python 文件夹下打开终端，输入“python3 ffm_cw.py”即可运行所提供的 ffm_cw.py 示例。

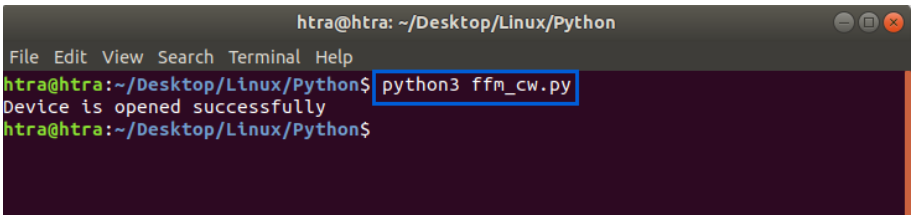


图 62 运行 python 范例

2.6.2 创建与编译新项目

在已按配置驱动文件章节，正确配置驱动文件的前提下，若想要创建 Python 项目，请参考以下流程：

1. 代码部分遵循 API 编程指南即可；

2. 运行程序时，将程序存放至范例文件夹（Python）下，按照 Python 范例使用章节流程执行即可。

3. 范例说明

范例默认使用标准射频输出端口（port = 0）；若设备已安装并启用中功率选件，且需要从中功率射频输出端口发射，请将 port 设置为 1。

3.1 发射连续波

ffm_cw：在单一频点上输出一个连续波，不涉及波形文件。可用于频点功能验证、射频链路通断测试等场景。

3.2 回放

ffm_playback：在固定频点回放程序生成的 IQ 波形，可使用波形生成函数或自定义生成 IQ 数据，适用于仿真波形回放等场景；

ffm_wav_playback：在固定频点回放一个或多个 IQ WAV 文件，多个文件可按顺序拼接后进行回放，适用于已有波形文件的回放测试；

ffm_multi_dev_sync：在固定频点实现多设备同步回放，通过参考时钟和触发信号实现设备间同步，适用于多设备同步信号输出等场景。

3.3 流

ffm_stream：实时读取并播放上位机波形文件或实时生成波形数据并播放，适用于大数据量场景。

3.4 动态更改配置

ffm_reconfig：在设备运行过程中更改配置，适用于动态频率功率调整、自动化测试等。

3.5 频率扫描

fscan_cw：连续波自动频率扫描，触发来临后，自动按照设定的步进和驻留时间进行频率扫描；

fscan_playback：使用仿真 IQ 波形进行频率扫描，可选择单步跳频或完整扫描两种触发行为；

fscan_wav_playback：使用单个 IQ WAV 文件进行频率扫描，每次触发切换至下一个频点，并在各频点持续回放预存波形；

fscan_mod：使用调制信号进行频率扫描，在各频点持续回放生成的调制波形；

fscan_trigger：设置不同触发源进行频率扫描，支持软件总线触发、外部上升沿触发、外部下降沿触发和 GNSS 秒脉冲触发，并可选择单步跳频或完整扫描。

3.6 功率扫描

lscan_cw：使用连续波信号进行功率扫描，可设置起始功率、终止功率、功率步进和驻留时间等参数；

lscan_playback：使用仿真 IQ 波形进行功率扫描，可选择单步功率切换或完整扫描两种触发行为；

lscan_trigger：设置不同触发源进行功率扫描，支持软件总线触发、外部上升沿触发、外部下降沿触发和 GNSS 秒脉冲触发，并可选择单步功率切换或完整扫描。

3.7 调制

digital_mod：生成数字调制信号，可配置数字调制方式、符号率、伪随机序列及成形滤波等参数；

am_mod：生成幅度调制信号，可配置调制速率、调制度和调制波形等参数；

fm_mod：生成频率调制信号，可配置调制速率、频偏和调制波形等参数；

pm_mod：生成相位调制信号，可配置调制速率、相位偏移和调制波形等参数；

dsss_mod：生成直接序列扩频信号，可配置符号率、数字调制方式、扩频码及成形滤波等参数；

ofdm_mod：生成正交频分复用信号，可配置 FFT 点数、符号数、保护载波、保护间隔及数字调制方式等参数；

pulse_mod：生成脉冲调制信号，可配置脉冲宽度和脉冲周期等参数；

multitone_mod：生成多音信号，可配置单音数量、频率间隔和相位模式等参数；

rampsweep_mod：生成线性扫频信号，可配置扫宽、扫描时间和周期等参数；

awgn_mod：生成高斯白噪声信号，可配置信号带宽和波形时长等参数。

www.harogic.cn
cninfo@harogic.com
+025-8330 5049