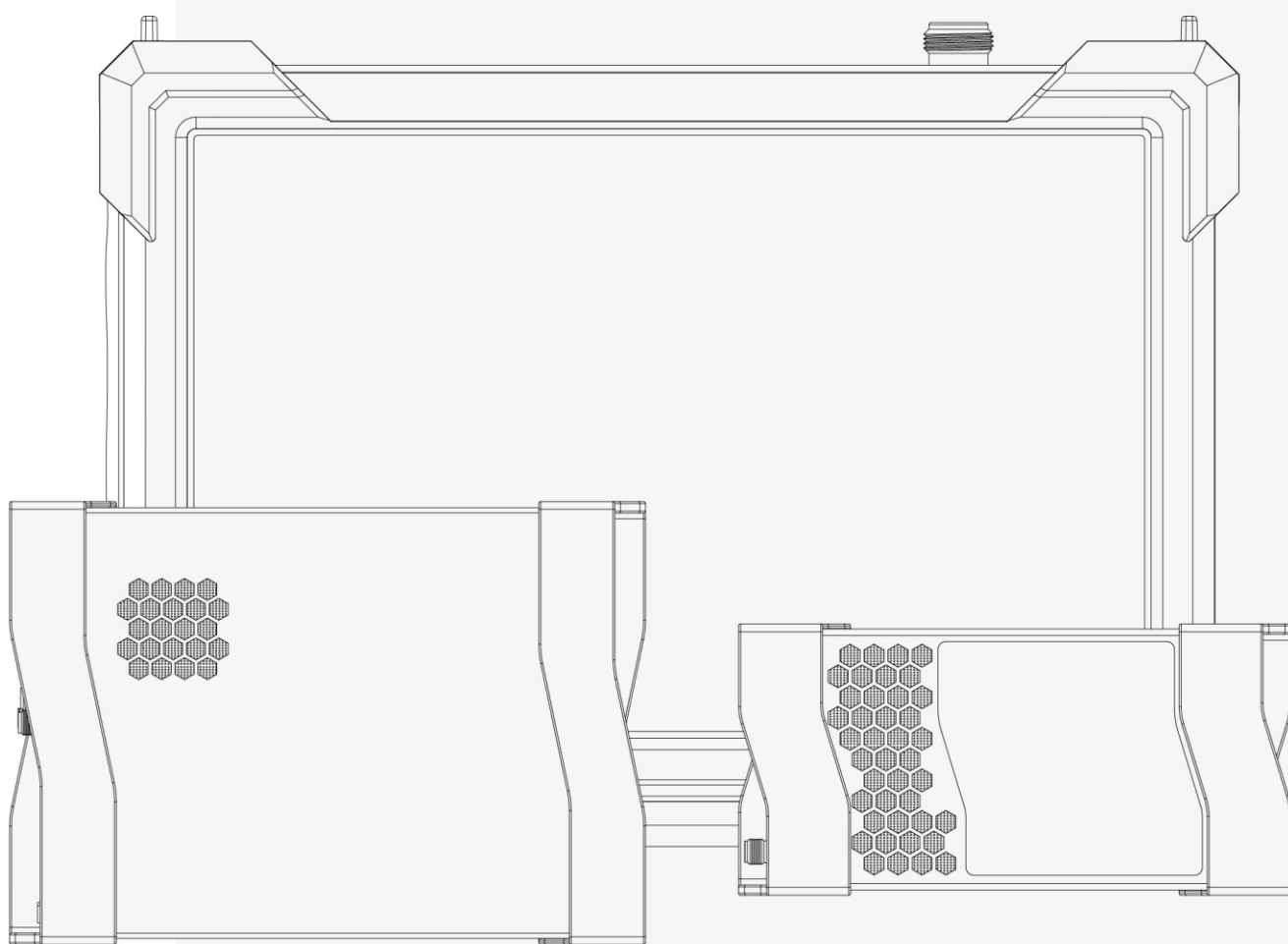




HTRA API 范例使用指南



目录

1. 版本管理	1
2. C/C++	2
2.1 配置开发环境	2
2.2 C++ 范例使用流程	4
2.2.1 常规 C++ 范例的使用	4
2.2.2 AM/FM 解调范例的使用	4
2.2.3 记录与读取范例的使用	5
2.3 Device 相关	6
2.3.1 获取设备信息	6
2.3.2 设备待机	6
2.3.3 GNSS 相关	6
2.3.4 获取和修改 NX 设备的 IP 地址	7
2.3.5 模式切换耗时	7
2.4 SWP 模式	7
2.4.1 标准频谱获取	7
2.4.2 最大和最小保持	7
2.4.3 迹线平均	7
2.4.4 自动配置测量	7
2.4.5 频率补偿	7
2.4.6 函数耗时、扫描速度和吞吐量	7
2.4.7 获取频谱峰值	7
2.4.8 信号和杂散	8
2.4.9 同时获取频谱和 IQ	8
2.4.10 使用 GNSS 的 10MHz 参考时钟	8
2.4.11 外部触发模式	8
2.4.12 迹线对齐方式	8
2.4.13 测试一定时间内可获取的频谱帧数	8
2.4.14 外触发标定内部 10MHz 参考时钟	8
2.4.15 相位噪声测量	8
2.4.16 信道功率测量	8
2.4.17 邻道功率比测量	8
2.4.18 百分比占用带宽测量	9
2.4.19 XdB 占用带宽测量	9
2.4.20 IM3 测量	9
2.4.21 频率间隔匹配 RBW	9
2.4.22 使用外部 10MHz 参考时钟	9
2.4.23 频谱数据记录	9
2.4.24 频谱数据读取	9
2.5 IQS 模式	9

2.5.1	获取固定点数或连续流的 IQ 数据	9
2.5.2	IQ 数据转化为电压 V 单位	9
2.5.3	下发配置和获取 IQ 的耗时	9
2.5.4	IQ 转频谱数据	10
2.5.5	IQ 转频谱（调用 liquid 库版本）	10
2.5.6	FM 解调播放	10
2.5.7	FM 解调数据分析	10
2.5.8	AM 解调播放	10
2.5.9	AM 解调数据分析	10
2.5.10	数字下变频	10
2.5.11	数字低通滤波器	10
2.5.12	音频分析	10
2.5.13	读取随寄软件的 IQS 流盘数据	11
2.5.14	将 IQ 数据记录为.wav 格式	11
2.5.15	.wav 换为.csv	11
2.5.16	流盘和读取 IQ 数据	11
2.5.17	多线程获取、处理和流盘 IQ 数据	11
2.5.18	GNSS1PPS 触发	11
2.5.19	IQS 多机同步	11
2.5.20	外触发	11
2.5.21	定时器触发	11
2.5.22	电平触发（预触发）	11
2.5.23	电平触发（触发延迟）	11
2.5.24	使用 GNSS 的 10MHz 参考时钟	12
2.6	DET 模式	12
2.6.1	获取固定点数或连续流的检波数据	12
2.6.2	检波数据记录	12
2.6.3	检波数据读取	12
2.7	RTA 模式	12
2.7.1	获取固定点数或连续流的实时频谱数据	12
2.7.2	RTA 模式每帧数据的耗时	12
2.7.3	实时频谱数据记录	12
2.7.4	实时频谱数据读取	12
2.8	ASG 信号源（选件）	12
2.8.1	输出单音/扫频/功率扫描信号	12
3.	Qt	13
3.1	配置开发环境	13
3.2	Qt 范例使用流程	15
3.3	Qt 范例说明	16
4.	Python	17
4.1	配置开发环境	17

4.2	Python 范例使用流程	18
4.3	Python 范例说明	18
4.3.1	获取设备信息	18
4.3.2	获取标准频谱数据	18
4.3.3	获取固定点数或时长的 IQ 数据	18
4.3.4	获取固定点数或时长的检波数据	19
4.3.5	获取固定点数或时长的实时频谱数据	19
4.3.6	IQ 转频谱数据	19
4.3.7	GNSS 相关	19
4.3.8	数字下变频	19
4.3.9	输出单音/扫频/功率扫描信号	19
5.	Matlab	20
5.1	运行 Matlab 范例	20
5.2	随寄范例简介	21
5.2.1	获取设备信息	21
5.2.2	获取标准频谱数据	21
5.2.3	创建多个游标，显示游标的频率和功率	21
5.2.4	每五分钟采集一次频谱的峰值	21
5.2.5	获取连续流或固定点数的 IQ 数据	21
5.2.6	将获取的 IQ 数据转为频谱数据	21
5.2.7	获取连续流或固定点数的检波数据	21
5.2.8	获取连续流或固定时长的实时频谱数据	21
5.2.9	获取连续流或固定时长的实时频谱数据并绘制	21
5.2.10	锁定 GNSS 天线和 DOCXO 晶振	22
5.2.11	多机同步	22
6.	C#	23
6.1	配置开发环境	23
6.1.1	开发环境确认	23
6.1.2	项目搭建	23
6.2	C#范例使用流程	26
6.3	C#范例说明	27
6.3.1	获取设备信息	27
6.3.2	获取标准频谱数据	27
6.3.3	获取固定点数或时长的 IQ 数据	27
6.3.4	获取固定点数或时长的检波数据	27
6.3.5	获取固定点数或时长的实时频谱数据	27
6.3.6	输出单音信号	27
6.3.7	IQ 转频谱数据	27
6.3.8	低通滤波	27
6.3.9	数字下变频	27
6.3.10	相位噪声测试	27

7. Labview	28
7.1 配置开发环境	28
7.1.1 在 Labview 环境中使用 API	28
7.2 Labview 范例使用流程	29
7.3 Labview 范例说明	30
7.3.1 获取设备信息	30
7.3.2 标准频谱获取	30
7.3.3 获取固定点数或时长的 IQ 数据	31
7.3.4 流盘和读取 IQ 数据	31
7.3.5 IQ 转频谱数据	31
7.3.6 数字下变频	31
7.3.7 音频分析	31
7.3.8 获取固定点数或时长的检波数据	31
7.3.9 获取固定点数或时长的实时频谱数据	31
7.3.10 ASG 信号源输出信号	31
8. Linux	32
8.1 环境版本适配性自查	32
8.2 随寄资料说明	33
8.2.1 HTRA_C++_Examples	33
8.2.2 HTRA_Qt_Examples	33
8.2.3 HTRA_Python_Examples	34
8.2.4 Install_HTRA_SDK	34
8.3 配置驱动文件	35
8.4 C++ 范例使用以及项目创建	36
8.4.1 C++ 范例使用	36
8.4.2 C++ 项目创建编译	37
8.4.3 C++ 项目交叉编译	41
8.5 Qt 范例使用以及项目创建	42
8.5.1 Qt 范例使用	42
8.5.2 Qt 项目创建编译	45
8.5.3 Qt 项目交叉编译	48
8.6 Python 范例使用以及项目创建	51
8.6.1 Python 范例使用	51
8.6.2 Python 项目创建	53

1. 版本管理

版本更新说明表

版本号	内容	时间
V2.1	1. C/C++中新增 检波数据记录 和 实时频谱数据记录 示例 2. 修改: 编译环境 要求	2026-08-05
V2.0	1. 初始版本	2025-12-05

2. C/C++

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

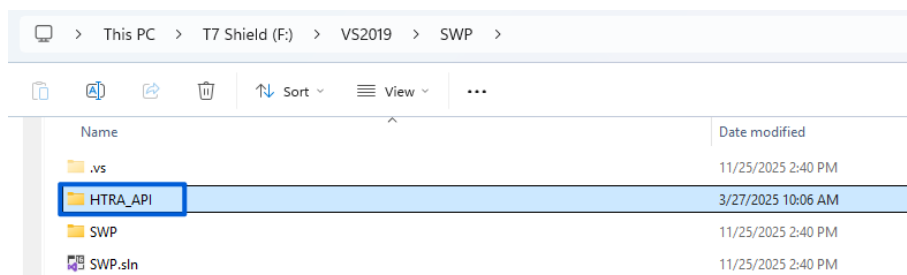


图 1 拷贝 HTRA_API 文件

3. 双击打开 SWP.sln，右击“源文件”->“添加”->“新建项”->“C++文件(.cpp)”，新建 SWP.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=..\HTRA_API\x86”；

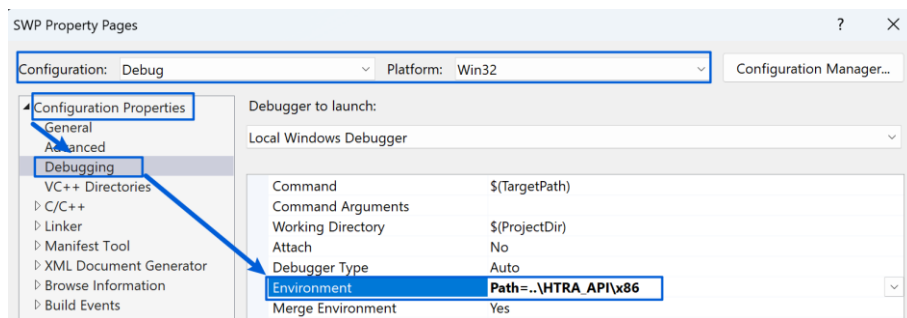


图 2 配置环境变量

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

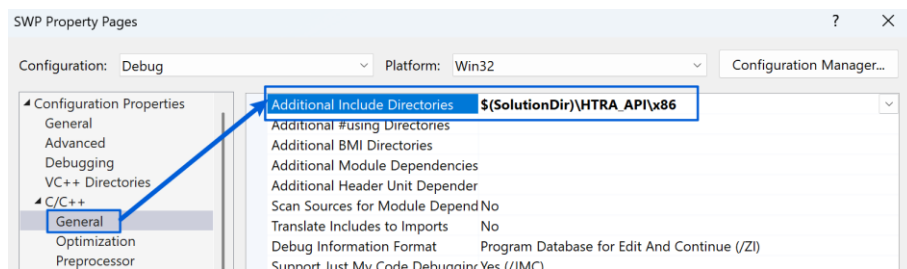


图 3 添加附加包含目录

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

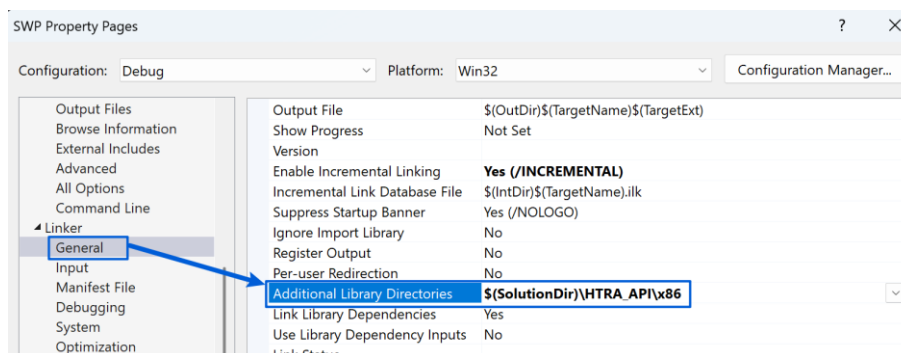


图 4 配置附加库目录

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

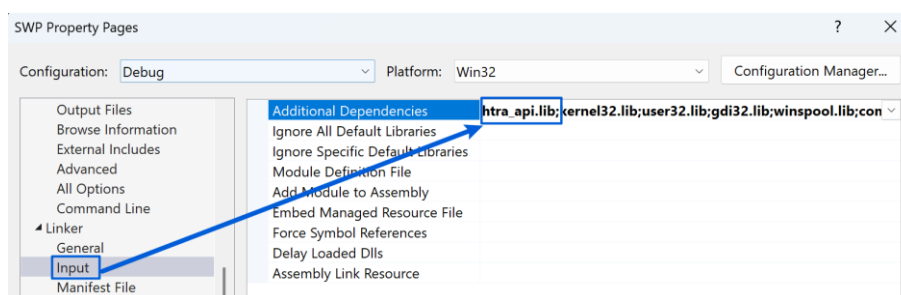


图 5 配置附加依赖

9. 至此，环境配置结束，可参照随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_H1P_C++_Examples”中的范例进行编程开发。

- 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“HTRA_API\x86”替换为“HTRA_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“HTRA_API\x86”替换为“HTRA_API\x64”

2.2 C++范例使用流程

注：同一时间只能运行一个范例，范例与 SASStudio4 不可同时运行。

2.2.1 常规 C++范例的使用

1. 使用 Visual Studio 打开随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\HTRA_C++_Examples”文件夹下解决方案 HTRA_C++_Examples.sln;
2. 点击右侧 HTRA_C++_Examples 项目，打开源文件中的 main.cpp 文件;
3. 每个示例程序都封装在单独的函数中，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。

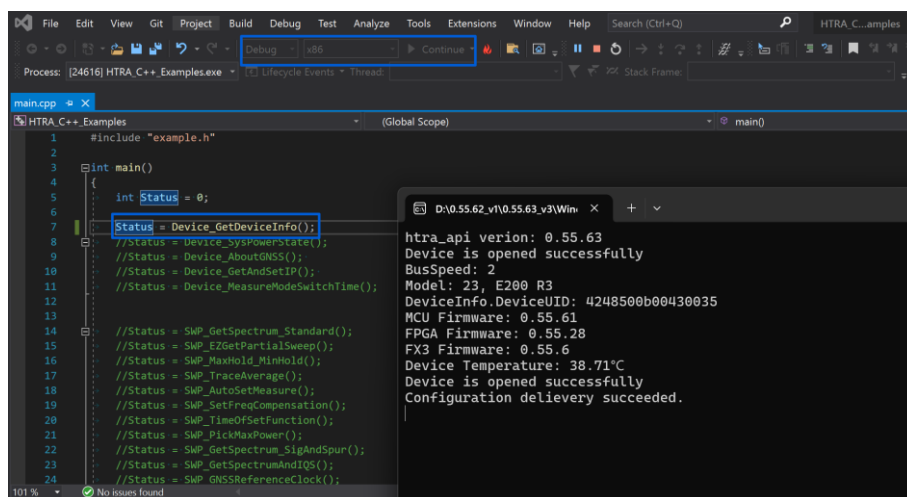


图 6 运行 Device_GetDeviceInfo 示例

2.2.2 AM/FM 解调范例的使用

1. 使用 Visual Studio 打开随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\Htra_Demodulation”文件夹下解决方案 Htra_Demodulation.sln;
2. 点击右侧 Htra_Demodulation 项目，打开源文件中的 main.cpp 文件;
3. C++随寄范例中每个例程都封装在单独的函数中，使用范例时取消对应例程的注释并保存，选择编译架构，点击运行。

2.2.3 记录与读取范例的使用

1. 使用 Visual Studio 打开随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_C++_Example s\Htra_RecordingandPlayBack”文件夹下的解决方案 Htra_RecordingandPlayBack.sln;
2. 双击右侧 Htra_RecordingandPlayBack 项目，打开源文件中的 main.cpp 文件;
3. 每个记录与读取的例程都封装在单独的函数中，使用时取消相应示例的注释即可。

■ 读取范例的使用

- 1) 例如读取 IQS 模式流盘数据时，将 main.cpp 中 IQSMODE_PlayBack 函数取消注释并保存;
- 2) 将 IQS 模式下的记录文件数据放至“Windows\HTRA_H1P_API_Example\HTRA_C++_Example s\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data”文件夹中;
- 3) 点击进入 IQSMODE_PlayBack.cpp，修改 IQSMODE_PlayBack()函数中记录文件的名称;

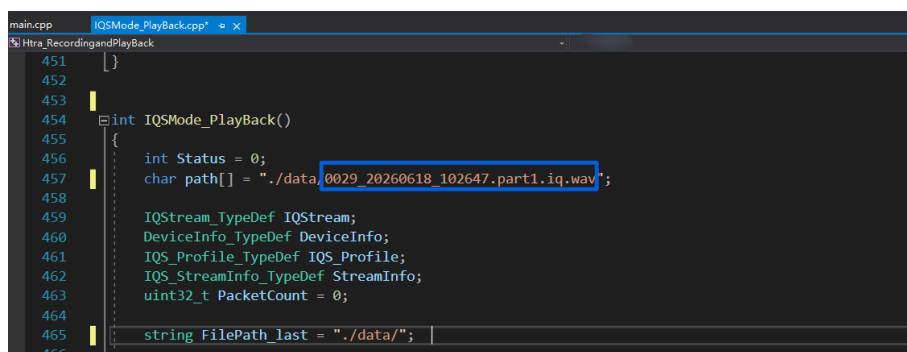


图 7 修改记录文件名称

- 4) 运行程序即可在 data 文件夹中得到 IQS 模式下记录文件的解析数据“IQSMODE_Data.txt”。

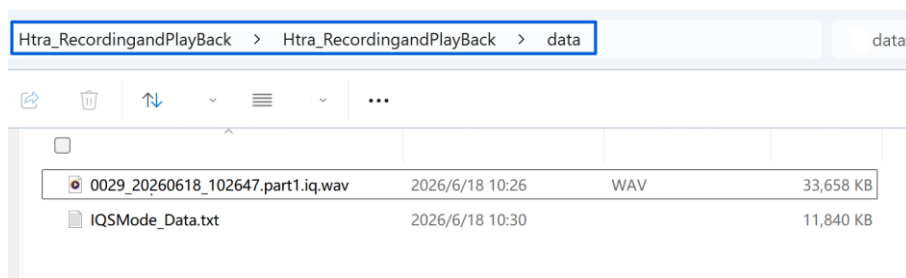


图 8 查看 IQS 模式下解析数据

■ 记录范例的使用

- 1) 例如测试使用 IQSMODE_Recording 例程，在 main.cpp 中取消注释并保存更改；
- 2) 点击进入 IQSMODE_Recording()函数，配置相关参数，然后运行程序，运行期间会持续记录文件，直至手动停止；
- 3) 可在“HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data”文件夹中得到 IQS 模式下的记录文件数据；

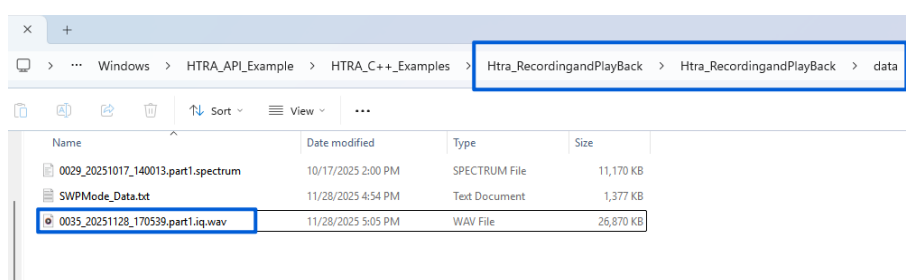


图 9 查看 IQ 流模式下记录文件数据

2.3 Device 相关

2.3.1 获取设备信息

Device_GetDeviceInfo.cpp: 获取设备信息，包括：API 版本、上位机与设备相连的 USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

2.3.2 设备待机

Device_SysPowerState.cpp: 设置设备待机状态的范例，可以设置为正常工作状态和射频处于下电状态（低功耗）。

2.3.3 GNSS 相关

Device_AboutGNSS.cpp: 获取 GNSS 模块获取的经纬度、海拔和时间等信息，获取 SWP 模式下 MeasAuxInfo 中 GNSS 相关的经纬度和时间信息，获取 IQS 模式下 IQStream.DeviceState 中的经纬度和时间信息。

2.3.4 获取和修改 NX 设备的 IP 地址

Device_GetAndSetIP.cpp: 获取设备 IP 地址，并通过设备 UID 或者设备当前的 IP 修改 IP 地址。

2.3.5 模式切换耗时

Device_MeasureModeSwitchTime.cpp: 获取当前上位机切换不同模式所需要的时间。

2.4 SWP 模式

2.4.1 标准频谱获取

SWP_GetSpectrum_Standard.cpp: 通过调用函数接口获取频谱数据。

2.4.2 最大和最小保持

SWP_MaxHold_MinHold.cpp : 将迹线模式设置为 MaxHold 或 MinHold , 并使用 SWP_ResetTraceHold 重置保持。

2.4.3 迹线平均

SWP_TraceAverage.cpp: 对获取迹线进行平均处理。

2.4.4 自动配置测量

SWP_AutoSetMeasure.cpp: 根据具体 SWP 应用, 自动配置相关参数, 通过下发自动配置参数, 完成测量。

2.4.5 频率补偿

SWP_SetFreqCompensation.cpp: 当存在外接衰减器时, 可对相应频段进行补偿, 使得测试结果依然准确。

2.4.6 函数耗时、扫描速度和吞吐量

SWP_TimeOfSetFunction.cpp : 获取 SWP_Configuration 、 SWP_GetPartialSweep 和 SWP_GetFullSweep 函数的调用耗时与当前配置下的扫描速度、吞吐量。

2.4.7 获取频谱峰值

SWP_PickMaxPower.cpp: 获取当前频谱的最大功率点与其对应的频率点。

2.4.8 信号和杂散

SWP_GetSpectrum_SigAndSpur.cpp: 可以在获取频谱数据后区分信号和杂散。

2.4.9 同时获取频谱和 IQ

SWP_GetSpectrumAndIQS.cpp: 同时获取频谱数据与 IQ 数据。

2.4.10 使用 GNSS 的 10MHz 参考时钟

SWP_GNSSReferenceClock.cpp : SWP 模式下使用高品质 GNSS 模块的 10MHz 参考时钟。

2.4.11 外部触发模式

SWP_GetSpectrum_Trigger.cpp: 获取触发源设置为外触发时的频谱数据。

2.4.12 迹线对齐方式

SWP_GetSpectrum_TraceAlign.cpp: 在迹线对齐方式为对齐至起始频率或对齐至中心频率时，获取频谱数据。

2.4.13 测试一定时间内可获取的频谱帧数

SWP_Fixedtime_GetFrames.cpp: 循环 50 次获取 10s 频谱数据，得出在 10s 内能获取的平均频谱帧数。

2.4.14 外触发标定内部 10MHz 参考时钟

SWP_CalibrateRefClock.cpp: 设备通过 GNSS-1PPS 或者通过 Ext 触发进行 Clock 校准的范例。

2.4.15 相位噪声测量

SWP_Meas_PhaseNoise.cpp: 测量接收信号在用户指定频偏处的单边带相位噪声 (单位: dBc/Hz), 支持多频点配置。

2.4.16 信道功率测量

SWP_Meas_ChannelPower.cpp: 测量接收信号的信道功率。

2.4.17 邻道功率比测量

SWP_Meas_ACPR.cpp: 测量接收信号的邻道功率比。

2.4.18 百分比占用带宽测量

SWP_Meas_OBW.cpp: 以百分比方式测量接收信号的占用带宽。

2.4.19 XdB 占用带宽测量

SWP_Meas_XdBBW.cpp: 以 XdB 方式测量接收信号的占用带宽。

2.4.20 IM3 测量

SWP_Meas_IM3.cpp: 接收信号的 IM3 测量。

2.4.21 频率间隔匹配 RBW

SWP_RBW_Spaced_Trace.cpp: 使迹线两点之间的频率间隔接近所设置的 RBW 值。

2.4.22 使用外部 10MHz 参考时钟

SWP_RefCLKSource_External.cpp: 使用外部 10MHz 参考时钟（以 SWP 模式下使用外部 10MHz 参考时钟为例，其他模式使用方法一致）。

2.4.23 频谱数据记录

SWPMode_Recording.cpp: 记录 SWP 模式下获取的频谱数据，生成.spectrum 格式文件。

2.4.24 频谱数据读取

SWPMode_PlayBack.cpp: 读取 SWP 模式下记录的.spectrum 文件，解析生成.txt 文档。

2.5 IQS 模式

2.5.1 获取固定点数或连续流的 IQ 数据

IQS_GetIQ_Standrad.cpp: 在专业配置下获取固定点数或连续流的 IQ 数据。

2.5.2 IQ 数据转化为电压 V 单位

IQS_ScaleIQDataToVolts.cpp: 将获取的 IQ 数据转换为以 V 为单位的数据。

2.5.3 下发配置和获取 IQ 的耗时

IQS_ConfigandGetIQ_Time.cpp: 获取 IQS_Configuration、IQS_GetIQStream_PM1 函数的调用耗时。

2.5.4 IQ 转频谱数据

DSP_IQSToSpectrum.cpp: 将获取到的时域 IQ 数据通过频谱分析方法转换为频谱数据。

2.5.5 IQ 转频谱 (调用 liquid 库版本)

IQS_ToSpectrumByLiquid.cpp: 调用 liquid 库将获取到的时域 IQ 数据通过频谱分析方法转换为频谱数据。

2.5.6 FM 解调播放

DSP_FMDemod.cpp: 对 IQ 数据进行 FM 解调, 并播放解调后的音频。

2.5.7 FM 解调数据分析

IQS_FMDataAnalysis.cpp: 对 FM 解调之后的 IQ 数据进行音频分析, 得到音频电压、音频频率、信纳德和总谐波失真。

2.5.8 AM 解调播放

DSP_AMDemod.cpp: 对 IQ 数据进行 AM 解调, 并播放解调后的音频。

2.5.9 AM 解调数据分析

IQS_AMDataAnalysis.cpp: 对 FM 解调之后的 IQ 数据进行音频分析, 得到音频电压、音频频率、信纳德和总谐波失真。

2.5.10 数字下变频

DSP_DDC.cpp: 对获得的 IQ 数据进行重采样。

2.5.11 数字低通滤波器

DSP_LPF.cpp: 对获得的 IQ 数据进行低通滤波。

2.5.12 音频分析

IQS_AudioAnalysis.cpp: 对解调之后的 IQ 数据进行音频分析, 得到音频电压、音频频率、信纳德和总谐波失真。

2.5.13 读取随寄软件的 IQS 流盘数据

IQSMode_PlayBack.cpp: 解析随寄软件在 IQS 模式下的记录文件数据, 并将读取的频谱数据写入 IQSMode_Data.txt 文件。

2.5.14 将 IQ 数据记录为.wav 格式

IQSMode_Recording.cpp: 将获得的 IQ 数据以.wav 的格式存储。

2.5.15 .wav 换为.csv

IQSMode_WavToCsv.cpp: 将 IQS 模式下的.wav 记录文件数据解析并提取 I 路和 Q 路数据, 转换为.CSV 格式文件进行保存。

2.5.16 流盘和读取 IQ 数据

IQS_GetIQToTxt.cpp: 将获得的 IQ 数据写入.txt 文件中。

2.5.17 多线程获取、处理和流盘 IQ 数据

IQS_Multithread_GetIQ_FFT_Write: 同时获取 IQ 数据、做 FFT 和将 IQ 数据写入.txt 文件。

2.5.18 GNSS1PPS 触发

IQS_GNSS_1PPS.cpp: 配置触发源为系统内 GNSS 提供的 1PPS 信号。

2.5.19 IQS 多机同步

IQS_MultiDevSync_fixed.cpp: 多台设备在同一时刻同时采集同一信号。

2.5.20 外触发

IQS_ExternalTrigger.cpp: 配置触发源为外部触发。

2.5.21 定时器触发

IQS_TimerTrigger.cpp: 配置触发源为定时器触发。

2.5.22 电平触发 (预触发)

IQS_LevelTrigger_PreTriggerr.cpp: 配置触发源为电平触发, 并设置预触发时间。

2.5.23 电平触发 (触发延迟)

IQS_LevelTrigger_TriggerDelay.cpp: 配置触发源为电平触发, 并设置触发延迟时间。

2.5.24 使用 GNSS 的 10MHz 参考时钟

IQS_Enable_GNSS_10MHz.cpp: 在 IQS 模式下使用高品质 GNSS 模块的 10MHz 参考时钟。

2.6 DET 模式

2.6.1 获取固定点数或连续流的检波数据

DETMMode_Standard.cpp: 获取固定点数或连续流的检波数据。

2.6.2 检波数据记录

DETMMode_Recording.cpp: 记录 DET 模式下获取的功率流数据，并生成.det.wav 格式文件。

2.6.3 检波数据读取

DETMMode_PlayBack.cpp: 读取 DET 模式下记录的.det.wav 文件，并解析生成.txt 文档。

2.7 RTA 模式

2.7.1 获取固定点数或连续流的实时频谱数据

RTAMode_Standard.cpp: 获取固定点数（时长）或连续流的实时频谱数据。

2.7.2 RTA 模式每帧数据的耗时

RTAMode_Standard_perframe.cpp: 获取 100 帧数据，并计算每帧的平均处理时间。

2.7.3 实时频谱数据记录

RTAMode_Recording.cpp: 记录 RTA 模式下获取的实时频谱数据，并生成.rtspectrum 格式文件。

2.7.4 实时频谱数据读取

RTAMode_PlayBack.cpp: 读取 RTA 模式下记录的.rtspectrum 文件，并解析生成.txt 文档。

2.8 ASG 信号源（选件）

2.8.1 输出单音/扫频/功率扫描信号

ASG_SignalOutput.cpp: 按需输出单音/扫频/功率扫描信号。

3. Qt

3.1 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\HTRA_H1P_API\x64\htra_api”文件夹拷贝至创建的 QtTest 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”；

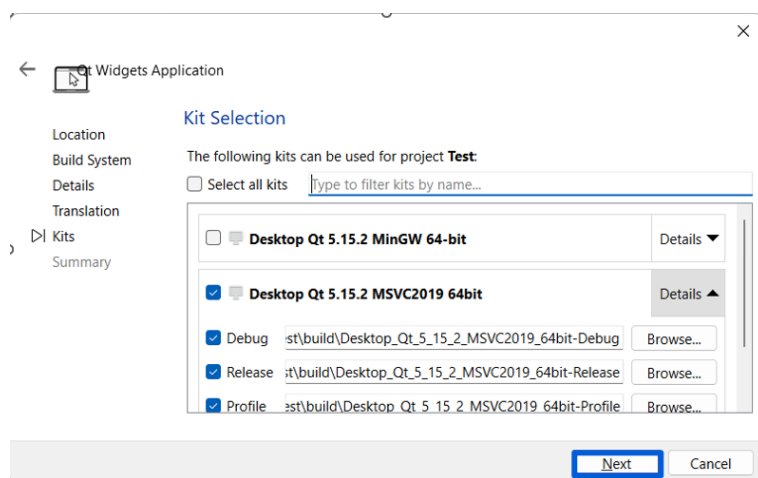


图 10 选择构建环境

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\htra_api”中的 htra_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；

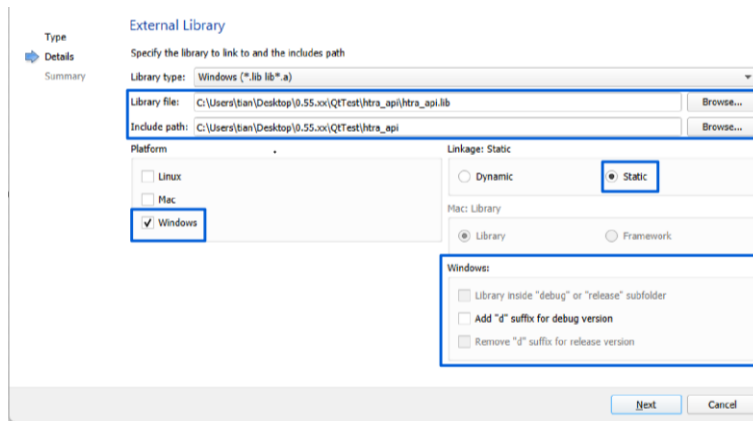


图 11 设置构建配置

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_TARGETDEPS+=\$\$\$PWD/../../htra_api/libhtra_api.a”代码，保存后可正常编写项目代码。

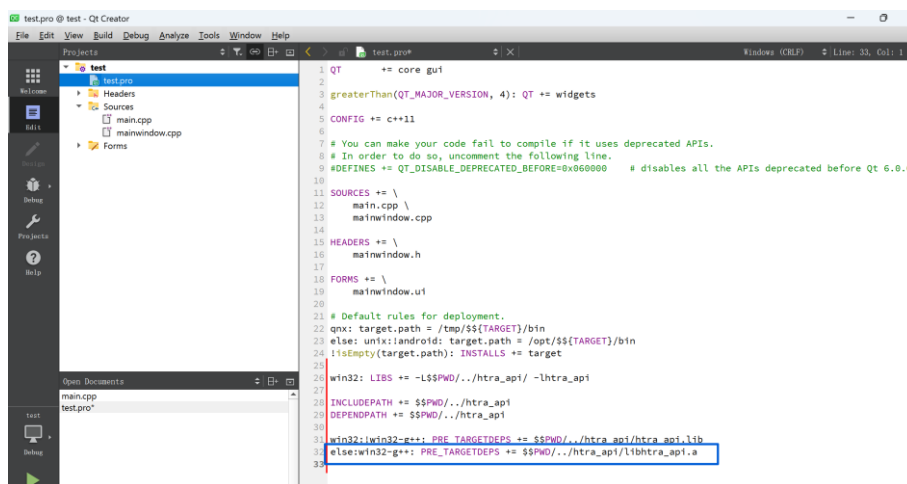


图 12 修改 test.pro 文件

12. 编写完代码后，点击运行。程序正确运行示意如下所示。

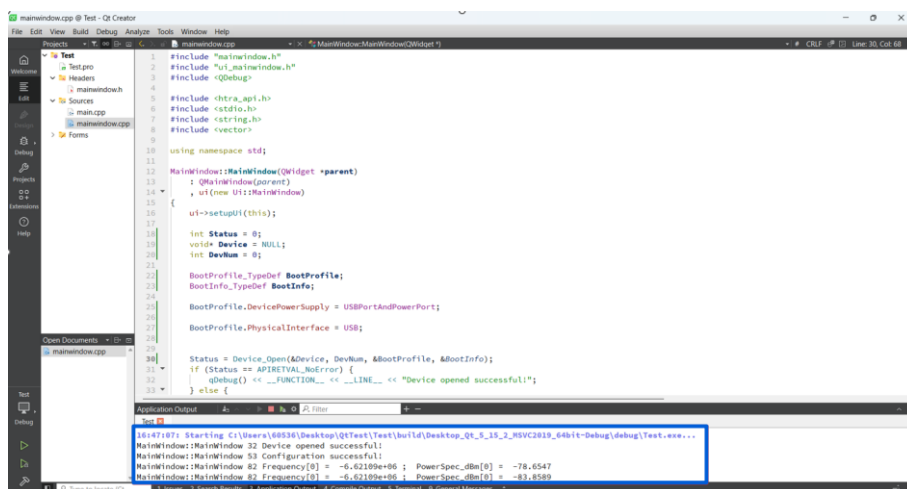


图 13 正确运行示意

3.2 Qt 范例使用流程

注：同一时间只能运行一个范例，范例与 SASstudio4 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 HTRA_Qt_Example 项目的使用为例：

1. 使用 Qt Creator 打开随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_Qt_Examples\HTRA_Qt_Example\HTRA_Qt_Example”文件夹下 HTRA_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“HTRA_Qt_Example\htra_api”文件夹中的库替换为“Windows\HTRA_H1P_API\X86”中 32 位库）；

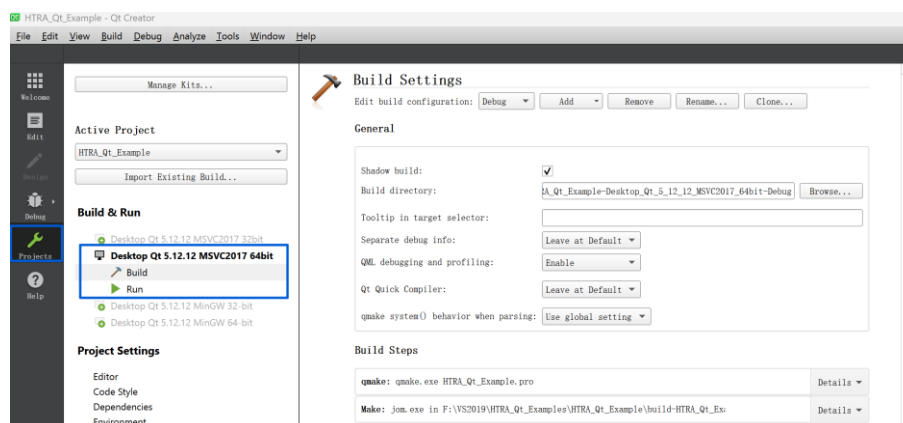


图 14 配置 64 位的构建环境

3. 环境配完后，点击“编辑”，打开“HTRA_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；

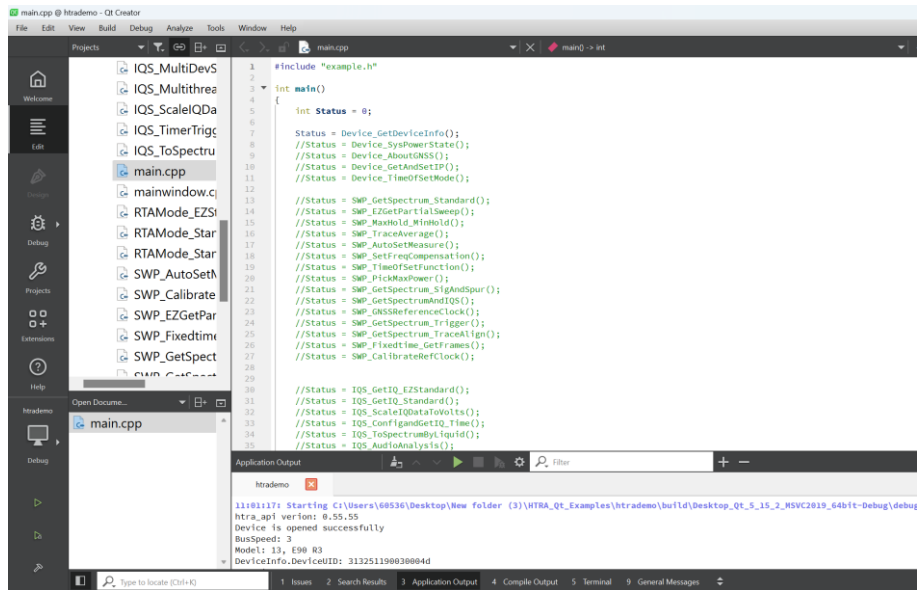


图 15 运行 QT 示例

3.3 Qt 范例说明

随寄 U 盘中 Qt 范例包含常规范例 (HTRA_Qt_Example), FM 与 AM 解调范例 (HTRA_Demodulation)

以及使用 liquid 库实现 IQ 转频谱范例 (IQS_ToSpectrumBuLiquid), 具体范例作用参考 C/C++ 范例章节即可。

4. Python

4.1 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“\HTRA_H1P_API_Example\HTRA_Python_Examples”, 将“HTRA_API”文件夹与“htra_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。

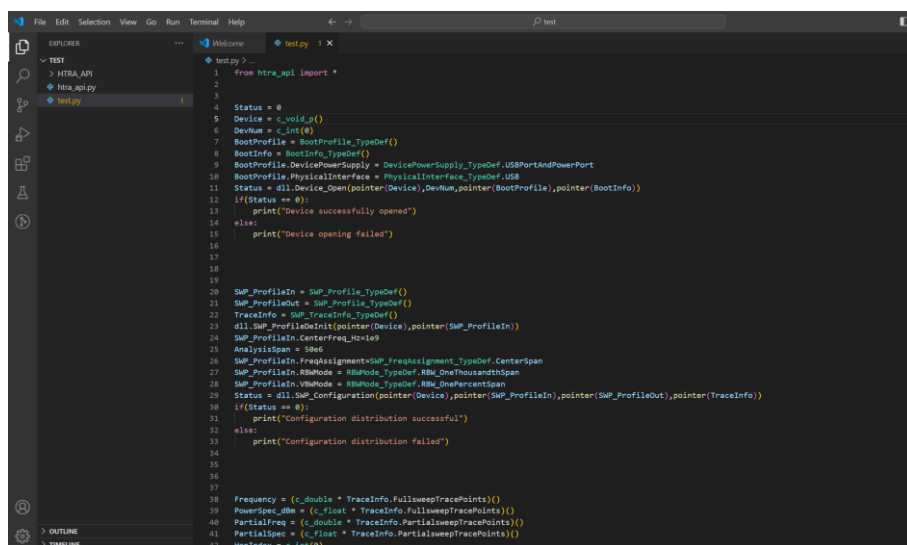


图 16 新建 Python 工程

4.2 Python 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\HTRA_H1P_API_Example\HTRA_Python_Examples”项目文件夹。其中 htra_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 SWPMode_Standard.py），直接运行该文件，程序运行成功示意如下所示：

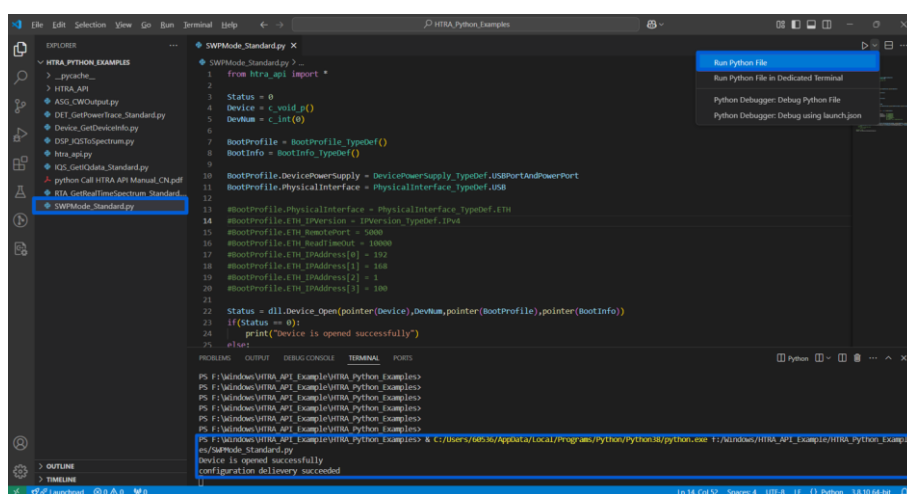


图 17 运行 SWPMode_Standard.py

4.3 Python 范例说明

4.3.1 获取设备信息

Device_GetDeviceInfo.py: 获取包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本以及设备温度在内的各种设备信息。

4.3.2 获取标准频谱数据

SWPMode_Standard.py: 获取指定频段内完整频谱数据(若上位机包含 matplotlib 库则绘制频谱图)。

4.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQdata_Standard.py: 在 IQS 模式的不同触发模式下获取 IQ 数据。

4.3.4 获取固定点数或时长的检波数据

DET_GetPowerTrace_Standard.py: 在 DET 模式的不同触发模式下获取功率检波数据（若上位机包含 matplotlib 库则绘制时间功率图）。

4.3.5 获取固定点数或时长的实时频谱数据

RTA_GetRealTimeSpectrum_Standard.py: 在 RTA 模式的不同触发模式下获取实时频谱数据（若上位机包含 matplotlib 库则绘制实时频谱图）。

4.3.6 IQ 转频谱数据

DSP_IQSToSpectrum.py: 将 IQS 模式下获取到的 IQ 数据转换成为频谱数据（若上位机包含 matplotlib 库则绘制频谱图与 IQ 时域图）。

4.3.7 GNSS 相关

Device_AboutGNSS.py: 获取 GNSS 模块获取的经纬度、海拔和时间等信息，获取 IQS 模式下 MeasAuxInfo 结构体中的经纬度和时间信息。

4.3.8 数字下变频

DSP_DDC.py: 对获得的 IQ 数据进行重采样，并绘制频谱图。

4.3.9 输出单音/扫频/功率扫描信号

ASG_SignalOutput.cpp: 按需输出单音/扫频/功率扫描信号。

5. Matlab

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

5.1 运行 Matlab 范例

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\HTRA_H1P_API_Example\HTRA_Matlab_Examples”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 SWPMode_Standard.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；

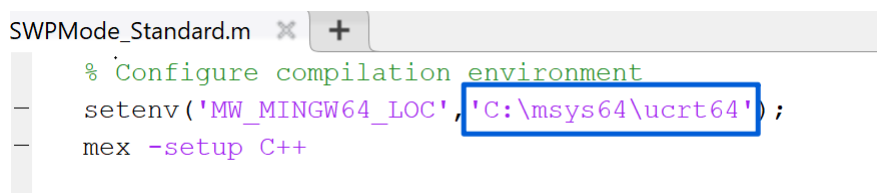


图 18 确认编译地址

4. 点击“Run”，出现 Figure1 窗口表示成功运行范例，各范例的功能说明，请参考随寄范例简介章节；

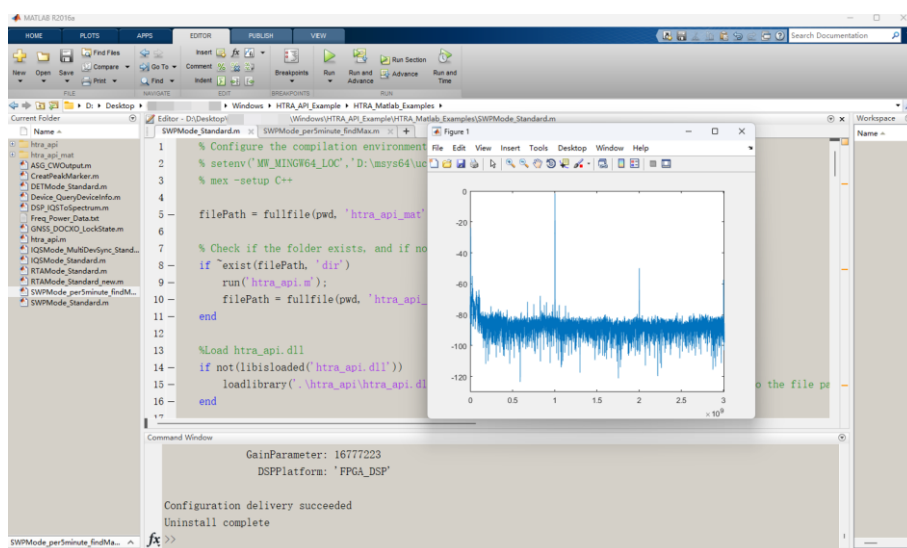


图 19 运行 SWPMode_Standard.m 文件

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。

5.2 随寄范例简介

5.2.1 获取设备信息

Device_QueryDeviceInfo.m: 获取设备信息, 包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

5.2.2 获取标准频谱数据

SWPMode_Standard.m: 获取指定频段内完整频谱数据。

5.2.3 创建多个游标, 显示游标的频率和功率

CreatPeakMarker.m: 获取指定频段内的频谱数据并创建游标并进行寻峰。

5.2.4 每五分钟采集一次频谱的峰值

SWPMode_per5minute_findMax.m: 获取指定频段内的频谱数据, 每五分钟在全局范围内寻找一次峰值。

5.2.5 获取连续流或固定点数的 IQ 数据

IQSMODE_Standard.m: 在 IQS 模式的不同触发模式下获取 IQ 数据。

5.2.6 将获取的 IQ 数据转为频谱数据

DSP_IQSToSpectrum.m: 获取 IQ 数据后, 将获取的 IQ 数据转为频谱数据。

5.2.7 获取连续流或固定点数的检波数据

DETMODE_Standard.m: 在 DET 模式的不同触发模式下获取功率检波数据。

5.2.8 获取连续流或固定时长的实时频谱数据

RTAMODE_Standard.m: 在 RTA 模式的不同触发模式下获取实时频谱数据。

5.2.9 获取连续流或固定时长的实时频谱数据并绘制

RTAMODE_Standard_3D.m: 在 RTA 模式下配置不同触发模式, 获取实时频谱数据, 并绘制实时频谱图及概率密度图。

5.2.10 锁定 GNSS 天线和 DOCXO 晶振

GNSS_DOCXO_LockState.m: 调用 API 接口锁定 GNSS 天线和 DOCXO 晶振, 仅适用于有 IO 拓展板的选件设备使用。

5.2.11 多机同步

IQSMultiDevSync_Standard.m: 在同参考时钟源输入和同触发源输入时, 两台设备同时获取 I/Q 数据, 并可查看其采集数据的同步性。

6. C#

6.1 配置开发环境

6.1.1 开发环境确认

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。

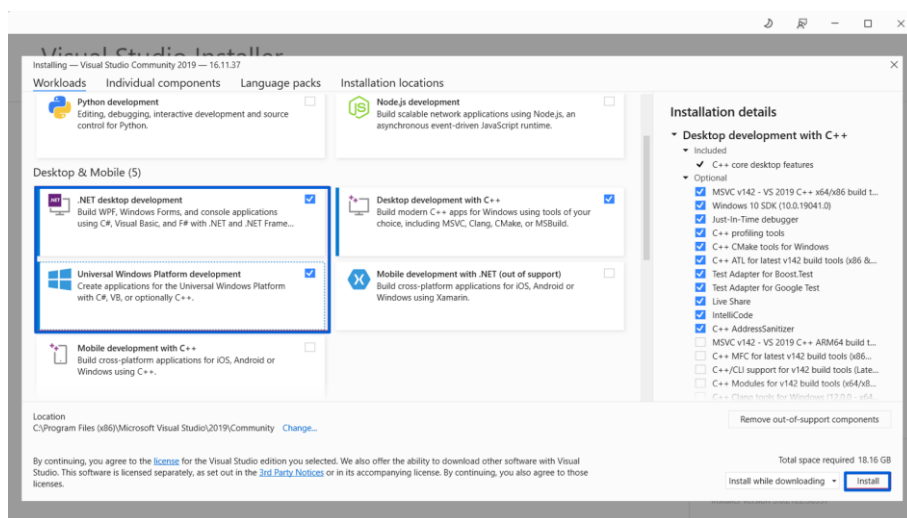


图 20 Visual Studio 中配置 C#开发环境

6.1.2 项目搭建

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”；

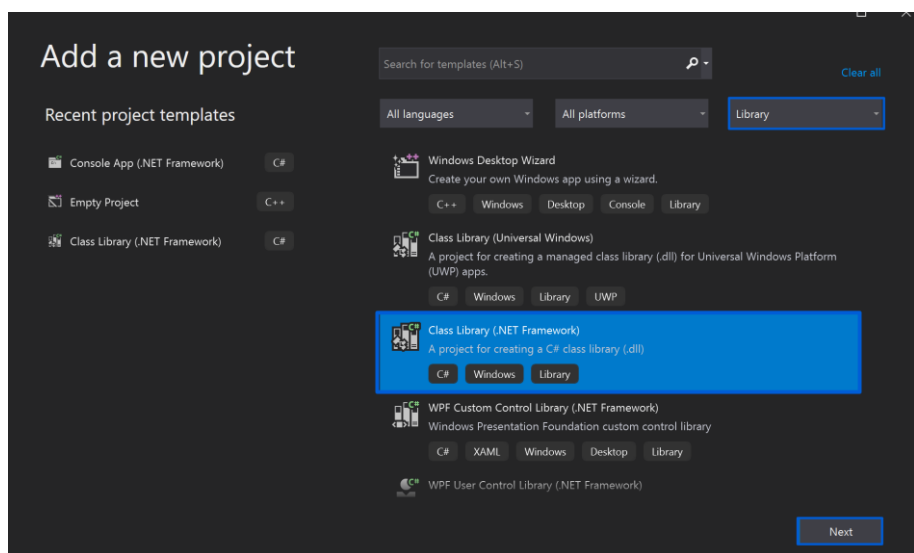


图 21 添加新项目

6. 按需求修改项目名称（如 HtraApi），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“htra_api\”，点击弹窗中的“OK”确认，最后保存并关闭属性窗口；

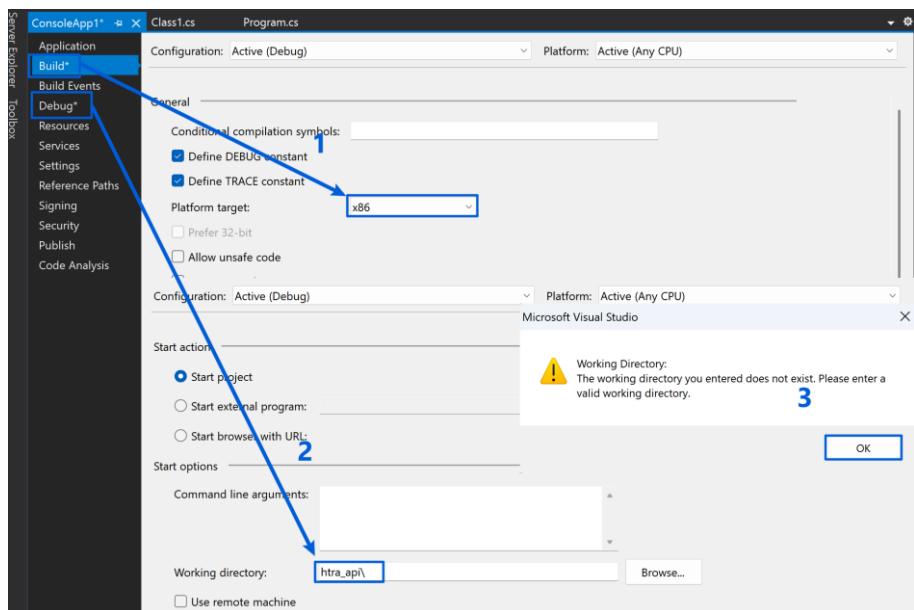


图 22 配置项目属性

8. 右击类库 HtraApi，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\HTRA_H1P_API_Example\HTRA_CSharp_Examples”文件夹下的 HtraApi.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“HtraApi”类库，点击“确定”，确认添加类库引用；

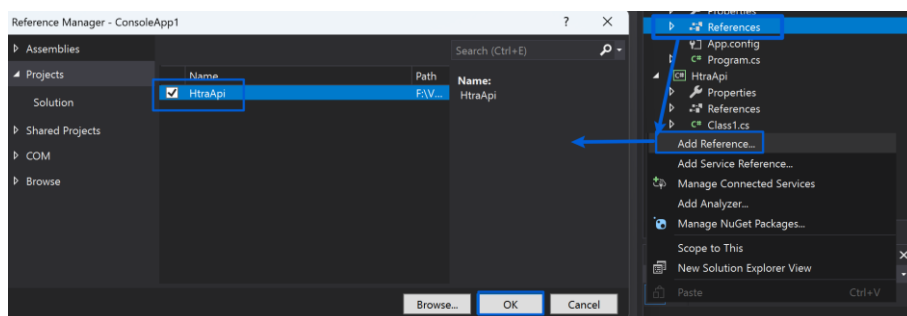


图 23 添加类库引用

11. 将随寄 U 盘中“\Windows\HTRA_H1P_API\x86”下的 htra_api 文件夹复制到项目文件夹的“bin\Debug”目录下，并确保“Htra_api\CalFile”文件夹中包含所用仪器的校准文件；

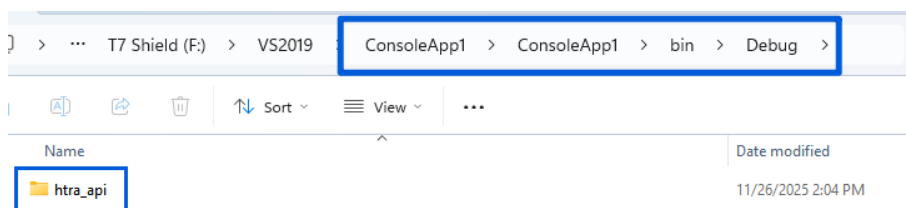


图 24 拷贝 htra_api 文件

12. 可参照随寄 U 盘中 C#范例，根据需要编写代码。

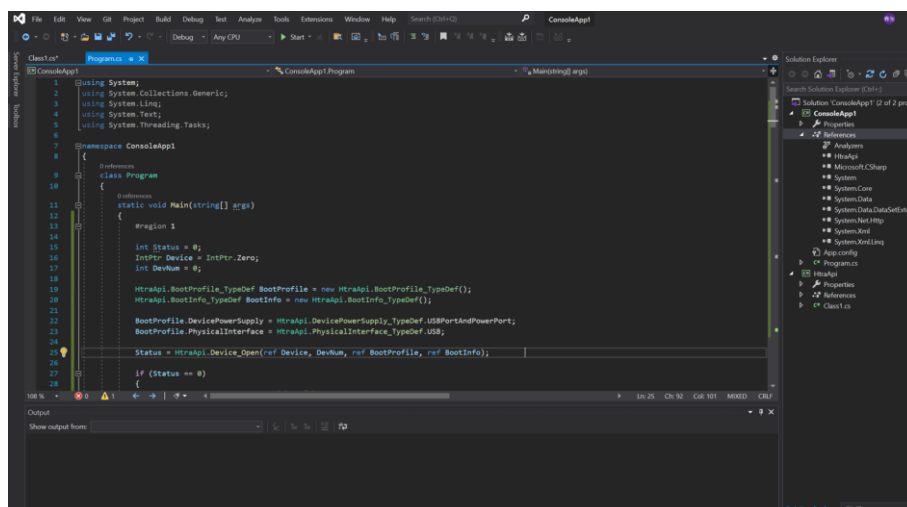


图 25 编写 C#示例

6.2 C#范例使用流程

1. 使用 Visual Studio 打开随寄 U 盘“Windows\HTRA_H1P_API_Example\HTRA_CSharp_Examples”文件夹下解决方案 HTRA_CSharp_Examples.sln;
2. 点击右侧 HTRA_CSharp_Examples 项目，点击其中的 Programe.cs 文件；

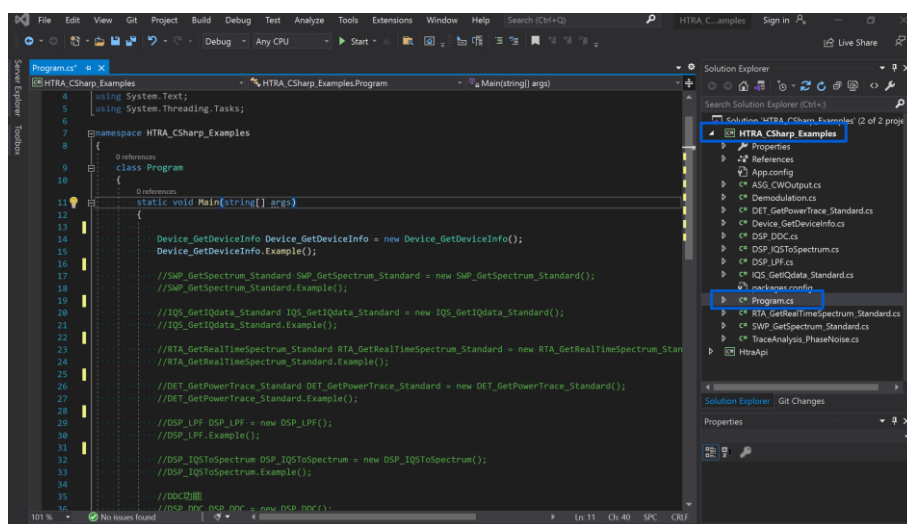


图 26 打开 Programe.cs 文件

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）。

例如使用 Device_GetDeviceInfo 例程时，取消注释并运行，程序运行成功示意如下所示。

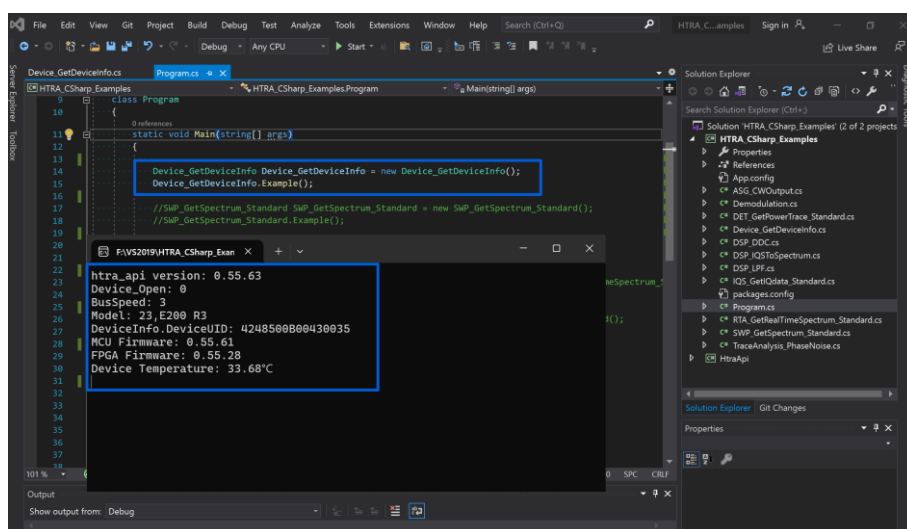


图 27 运行 Device_GetDeviceInfo.cs

6.3 C#范例说明

6.3.1 获取设备信息

Device_GetDeviceInfo.cs: 获取包括 API 版本、USB 版本、设备型号、设备 UID、MCU 版本、FPGA 版本以及设备温度在内的设备信息。

6.3.2 获取标准频谱数据

SWP_GetSpectrum_Standard.cs: 获取指定频段内完整频谱数据。

6.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQdata_Standard.cs: 在 IQS 模式的不同触发模式下获取 IQ 数据。

6.3.4 获取固定点数或时长的检波数据

DET_GetPowerTrace_Standard.cs: 在 DET 模式的不同触发模式下获取功率检波数据。

6.3.5 获取固定点数或时长的实时频谱数据

RTA_GetRealTimeSpectrum_Standard.cs: 在 RTA 模式的不同触发模式下获取实时频谱数据。

6.3.6 输出单音信号

ASG_CWOutput.cs: 含有信号源功能选件的设备通过 ASG 功能输出单音、频率扫描或功率扫描信号。

6.3.7 IQ 转频谱数据

DSP_IQSToSpectrum.cs: 将 IQS 模式下获取到的 IQ 数据转换成为频谱数据。

6.3.8 低通滤波

DSP_LPF.cs: 对获取到的 IQ 数据进行低通滤波并转频谱。

6.3.9 数字下变频

DSP_DDC.cs: 对获取到的 IQ 数据进行数字下变频并转频谱。

6.3.10 相位噪声测试

TraceAnalysis_PhaseNoise.cs: 相位噪声测试功能演示。

7. Labview

7.1 配置开发环境

7.1.1 在 Labview 环境中使用 API

1. 打开 Labview, 点击“创建项目”, 选择“项目”, 点击“完成”。之后会出现一个未命名的空项目, “Ctrl+S”保存项目, 选定项目保存路径并给项目命名(例如 HTRA_Labview_Examples), 然后点击“确定”;
2. 将 U 盘中“\Windows\HTRA_H1P_API_Example\HTRA_Labview_Examples\htra_api”文件夹拷贝到该项目的同级目录下。除此之外, 可以再创建一个 Examples 文件夹, 用于存放范例。如果有需要可以再创建一个文件夹如 Subvi, 用于存放子 vi;

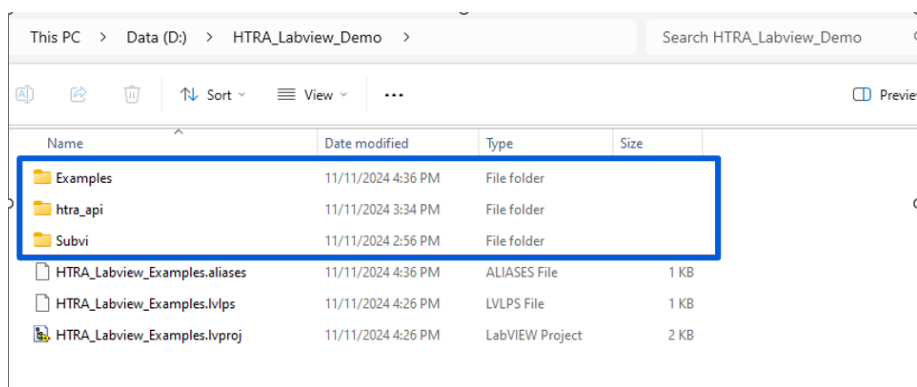


图 28 拷贝 htra_api

3. 在 Labview 工程中, 将新建的 Examples 文件夹和 htra_api 文件夹添加至工程中, 然后保存;

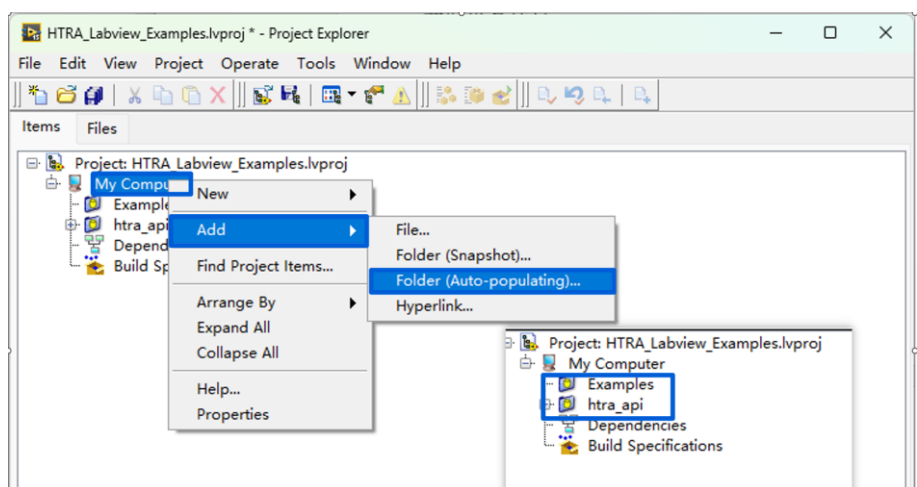


图 29 添加至工程

4. 在 Examples 文件夹中新创建 VI，在程序框图页内即可调用导出的 Labview API 函数，调用流程与 C 环境中一致；

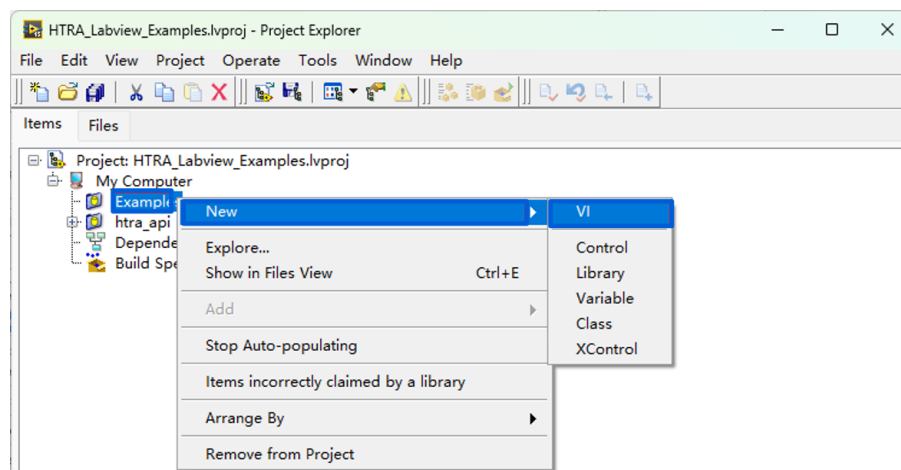


图 30 创建 VI

5. 最后将该程序保存至 Examples 文件夹，重新命名即可。

7.2 Labview 范例使用流程

随寄 U 盘中 Labview 范例使用流程如下：

1. 使用 Labview 打开随寄 U 盘中“Windows\HTRA_H1P_API_Example\HTRA_Labview_Examples”文件夹下的“HTRA_Labview_Examples.lvproj”工程；

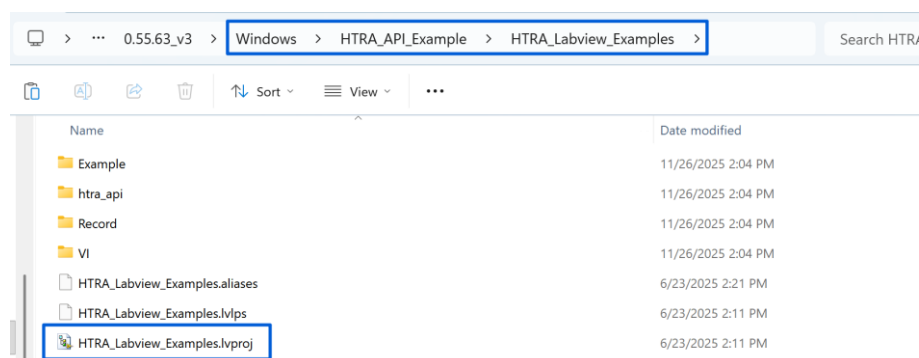


图 31 打开 Labview 工程

2. 双击打开 Example 文件夹中任意例程 VI，此处以 SWP_GetSpectrum_Standard.vi 为例；

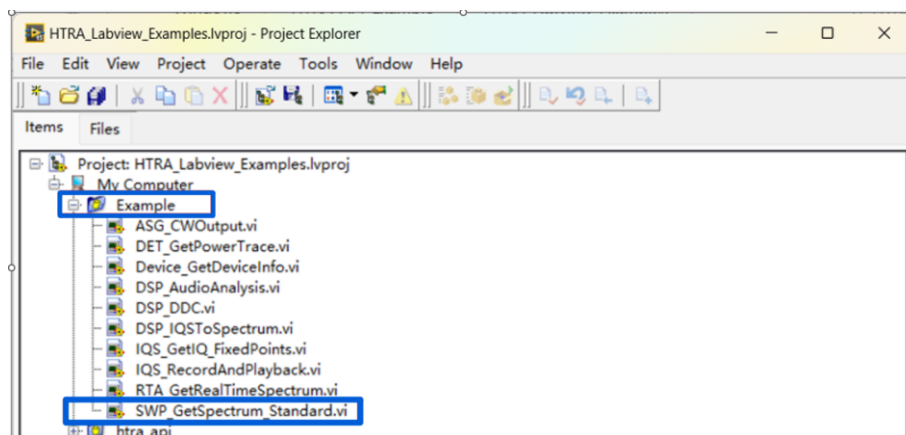


图 32 打开示例

3. 程序打开后，点击左上角的运行按钮，程序即可正常运行，运行界面如下所示：

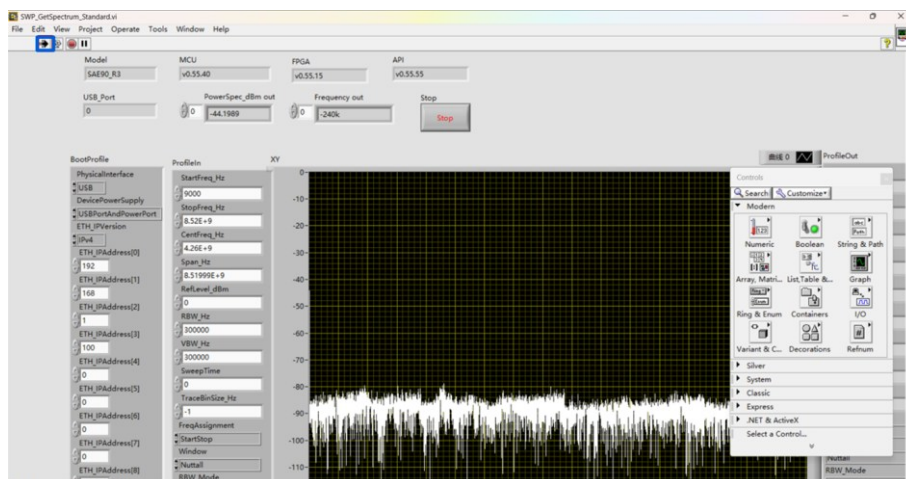


图 33 正确运行 SWP_GetSpectrum_Standard 示例

7.3 Labview 范例说明

7.3.1 获取设备信息

Device_GetDeviceInfo.vi: 获取各种设备信息的范例，包括：API 版本、设备型号、设备 UID、MCU 版本、FPGA 版本和设备温度。

7.3.2 标准频谱获取

SWP_GetSpectrum_Standard.vi: 获取指定频段内的标准频谱数据并显示图像。

7.3.3 获取固定点数或时长的 IQ 数据

IQS_GetIQ_Standard.vi: 获取固定点数的 IQ 数据, 当设备接收到 Bus 触发信号后, 设备返回固定点数的 IQ 数据。

7.3.4 流盘和读取 IQ 数据

IQS_RecordAndPlayback.vi: 获取 IQ 数据、将 IQ 数据记录为 txt 文件以及回放记录的 IQ 数据。

7.3.5 IQ 转频谱数据

DSP_IQSToSpectrum.vi: 获取 IQ 数据并将获取的 IQ 数据转为频谱数据。

7.3.6 数字下变频

DSP_DDC.vi: 对 IQ 数据流进行数字下变频并重采样生成子 IQ 流进行进一步的频谱分析。

7.3.7 音频分析

DSP_AudioAnalysis.vi: 分析音频的音频电压(V)、音频频率(Hz)、信纳德(dB)和总谐波失真(%)。

7.3.8 获取固定点数或时长的检波数据

DETMMode_Standard.vi: 获取固定点数的 DET 数据。当设备接收到 Bus 触发信号后, 设备返回固定点数的 DET 数据。

7.3.9 获取固定点数或时长的实时频谱数据

RTAMode_Standard.vi: 获取固定点数的 RTA 数据。当设备接收到 Bus 触发信号后, 设备返回固定点数的 RTA 数据。

7.3.10 ASG 信号源输出信号

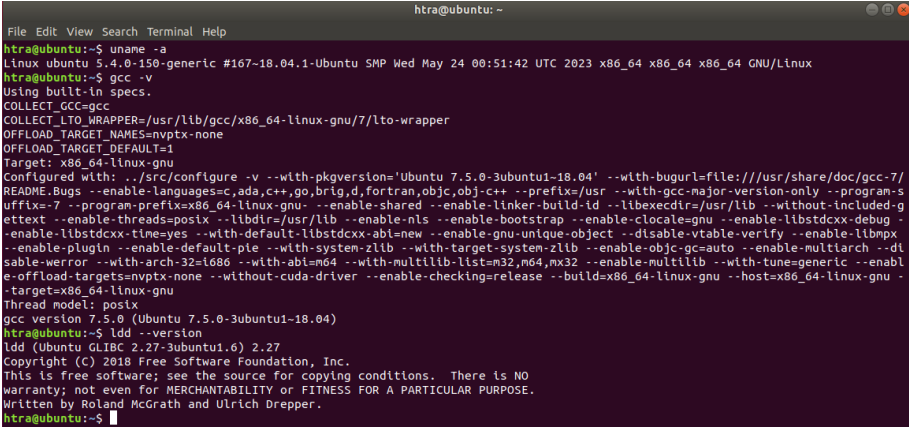
ASG_SignalOutput.vi: 控制设备内部信号发生器输出单音信号、扫频信号和功率扫描信号。

8. Linux

8.1 环境版本适配性自查

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）。

A terminal window titled 'htra@ubuntu: ~' showing the output of three commands. The first command 'uname -a' returns 'Linux ubuntu 5.4.0-150-generic #167-18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 GNU/Linux'. The second command 'gcc -v' shows the GCC version 7.5.0 and various configuration options. The third command 'ldd --version' returns 'ldd (Ubuntu GLIBC 2.27-3ubuntu1.6) 2.27'.

```
htra@ubuntu: ~  
htra@ubuntu:~$ uname -a  
Linux ubuntu 5.4.0-150-generic #167-18.04.1-Ubuntu SMP Wed May 24 00:51:42 UTC 2023 x86_64 x86_64 GNU/Linux  
htra@ubuntu:~$ gcc -v  
Using built-in specs.  
COLLECT_GCC=gcc  
COLLECT_LTO_WRAPPER=/usr/lib/gcc/x86_64-linux-gnu/7/lto-wrapper  
OFFLOAD_TARGET_NAMES=mvptx-none  
OFFLOAD_TARGET_DEFAULT=1  
Target: x86_64-linux-gnu  
Configured with: ../src/configure -v --with-pkgversion='Ubuntu 7.5.0-3ubuntu1-18.04' --with-bugurl=file:///usr/share/doc/gcc-7/README.Bugs --enable-languages=c,ada,c++,go,brig,d,fortran,objc,obj-c++ --prefix=/usr --with-gcc-major-version-only --program-suffix=-7 --program-prefix=x86_64-linux-gnu- --enable-shared --enable-linker-build-id --libexecdir=/usr/lib --without-included-g  
ettext --enable-threads=posix --libdir=/usr/lib --enable-nls --enable-bootstrap --enable-clocale=gnu --enable-libstdc++-debug -  
--enable-libstdc++-time=yes --with-default-libstdc++-abi=new --enable-gnu-unique-object --disable-vtable-verify --enable-libmpx  
--enable-plugin --enable-default-pie --with-system-zlib --with-target-system-zlib --enable-objc-gc=auto --enable-multitarch --dt  
sable-werror --with-arch=32-i686 --with-abi=m64 --with-multilib-list=m32,m64,mx32 --enable-multilib --with-tune=generic --enab  
le-offload-targets=mvptx:none --without-cuda-driver --enable-checking=release --build=x86_64-linux-gnu --host=x86_64-linux-gnu -  
-target=x86_64-linux-gnu  
Thread model: posix  
gcc version 7.5.0 (Ubuntu 7.5.0-3ubuntu1-18.04)  
htra@ubuntu:~$ ldd --version  
ldd (Ubuntu GLIBC 2.27-3ubuntu1.6) 2.27  
Copyright (C) 2018 Free Software Foundation, Inc.  
This is free software; see the source for copying conditions. There is NO  
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  
Written by Roland McGrath and Ulrich Drepper.  
htra@ubuntu:~$
```

图 34 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	Gcc7.5、glibc2.27 及以上
发行版	树莓派 4b 定制系统、ubuntu 18.04 等

8.2 随寄资料说明

目前随寄 U 盘 Linux 部分包含以下资料：

8.2.1 HTRA_C++_Examples

HTRA_C++_Examples 文件夹包含：

1. Examples 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：用于存放设备校准文件以及范例程序构建生成的可执行文件。

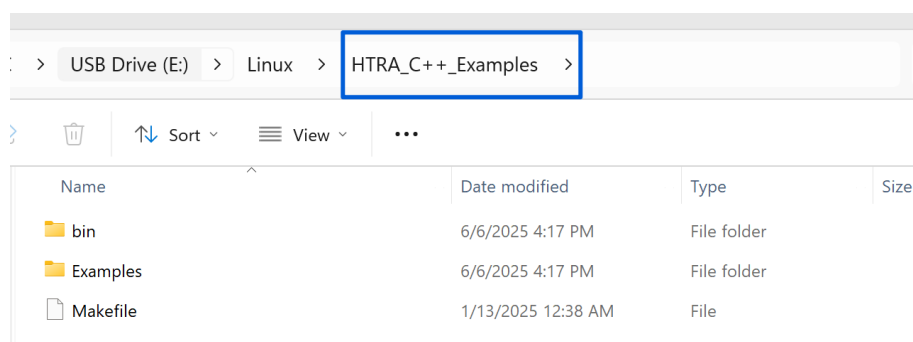


图 35 Linux 下 HTRA_C++_Examples 文件夹内容

8.2.2 HTRA_Qt_Examples

HTRA_Qt_Examples 文件夹包含：

1. htrademo 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：用于存放设备校准文件以及范例程序编译生成的可执行文件；
3. htraapi 文件夹：用于存放动态链接库。

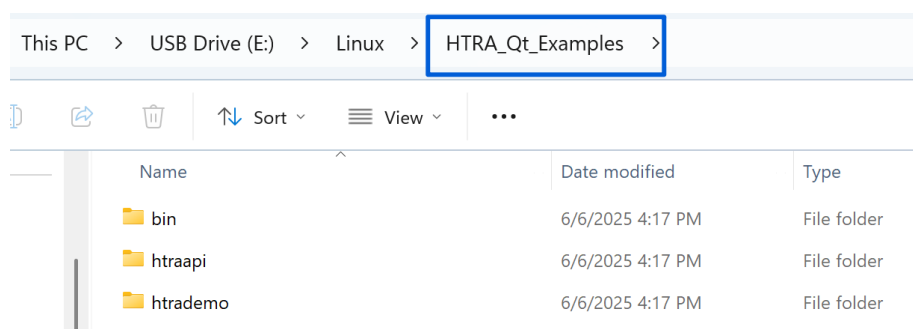


图 36 Linux 下 HTRA_Qt_Examples 文件夹内容

8.2.3 HTRA_Python_Examples

HTRA_Python_Examples 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：存放设备校准文件。
3. Htraapi 文件夹：用于存放动态链接库。

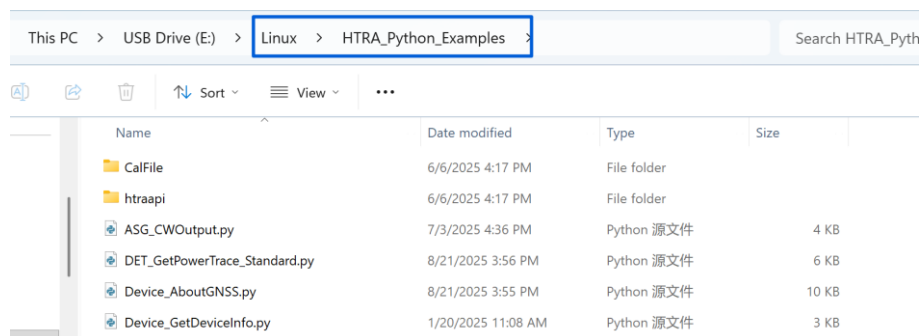


图 37 Linux 下 HTRA_Python_Examples 文件夹内容

8.2.4 Install_HTRA_SDK

1. Install_HTRA_SDK 文件夹包含：

install_htraapi_lib.sh：驱动配置脚本

htraapi 文件夹：存放驱动、头文件和库

2. Install_HTRA_SDK/htraapi 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_HTRA_SDK/htraapi/lib 文件夹中包含：

arrch64 文件夹：arrch64 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

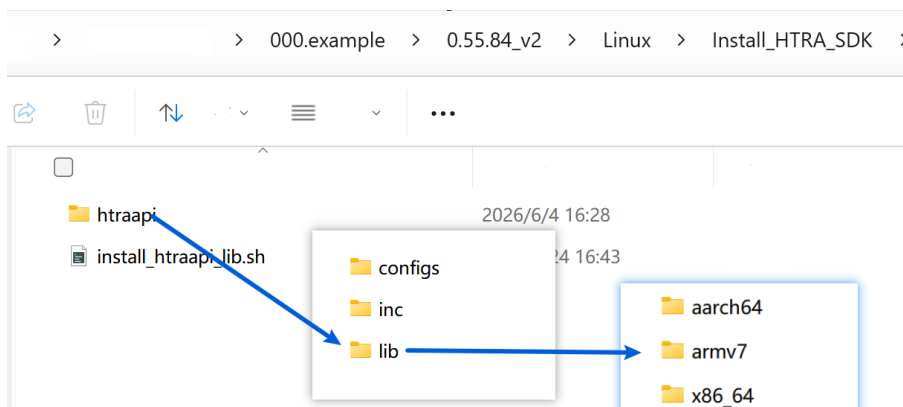


图 38 Linux 下 Install_HTRA_SDK 文件夹内容

8.3 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. 配置驱动文件：首先将 Install_HTRA_SDK 文件夹拖入 Linux 上位机，之后在 Install_HTRA_SDK 文件夹下打开终端，输入“sudo sh install_htraapi_lib.sh”配置驱动文件；

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

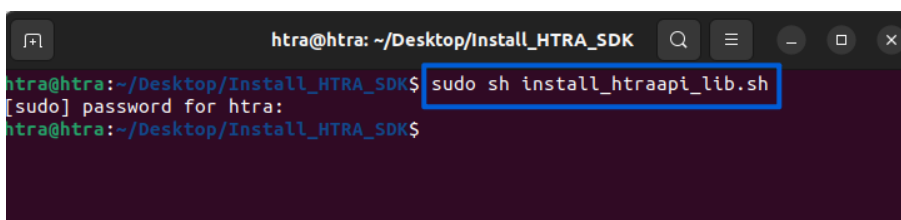


图 39 Linux 下配置驱动文件

2. 配置情况检查：正确连接设备与上位机（若上位机是虚拟机，需保证设备连接在虚拟机并保证 USB 兼容调整为 USB 3.1）并为设备正常供电，此时如图所示在终端输入“lsusb”，查看本机 USB 设备列表，其中“ID: 6430（或 ID: 3675 或 ID: 04b5 或 ID: 367f）”即为成功接入设备。

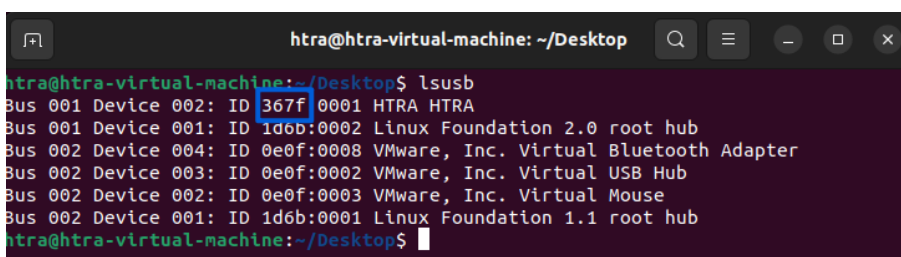


图 40 查看 Linux 下设备连接情况

8.4 C++范例使用以及项目创建

8.4.1 C++范例使用

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。

可参考以下流程，使用随寄 U 盘中 C++范例，（范例具体作用可直接查看 [C/C++章节](#)中相关描述）：

1. 选择需要编译的程序：首先将随寄 U 盘“Linux\HTRA_C++_Examples”文件夹拷贝至上位机。双击

打开 Examples 文件夹下 main.cpp，将需要测试使用的范例注释取消；

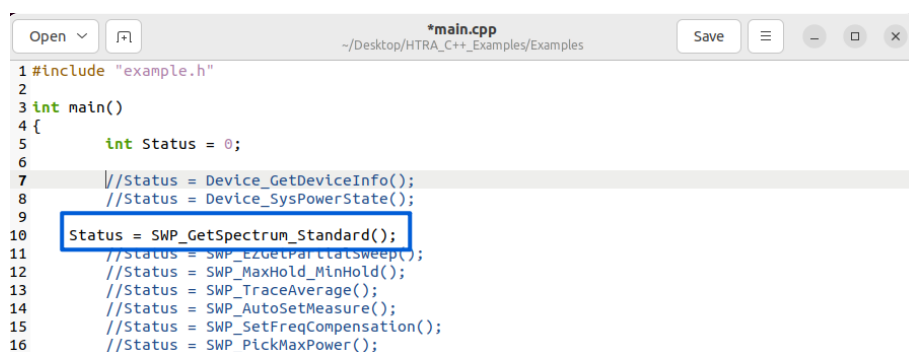


图 41 取消 SWP_GetSpectrum_Standard 范例注释

2. 编译所选示例：按[环境版本适配性自查章节](#)查看系统架构，在 HTRA_C++_Examples 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 SWP_GetSpectrum_Standard 例程为例）；

x86_64 系统输入：make Example=SWP_GetSpectrum_Standard

aarch64 系统输入：make TARG=aarch64 Example=SWP_GetSpectrum_Standard

armv7 系统输入：make TARG=armv7 Example=SWP_GetSpectrum_Standard

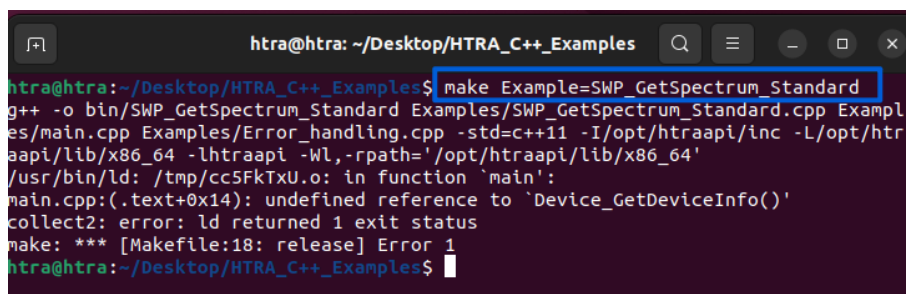


图 42 x86_64 系统下编译 SWP_GetSpectrum_Standard

3. 核对校准文件：进入“HTRA_C++_Examples\bin\CalFile”文件夹，确保文件夹中包含所用设备的校准文件（SAN-45 R6，SAN-60 R6，SAN-90 R0 设备跳过此步骤）；

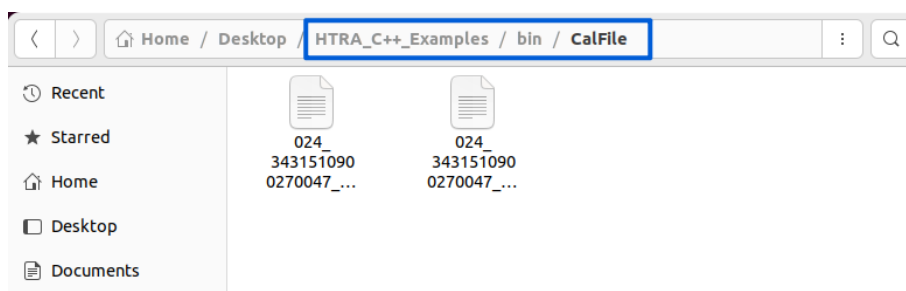


图 43 核对校准文件

4. 运行程序：在程序编译成功并确认校准无误后打开终端输入：

“./bin/SWP_GetSpectrum_Standard”，运行所选范例。

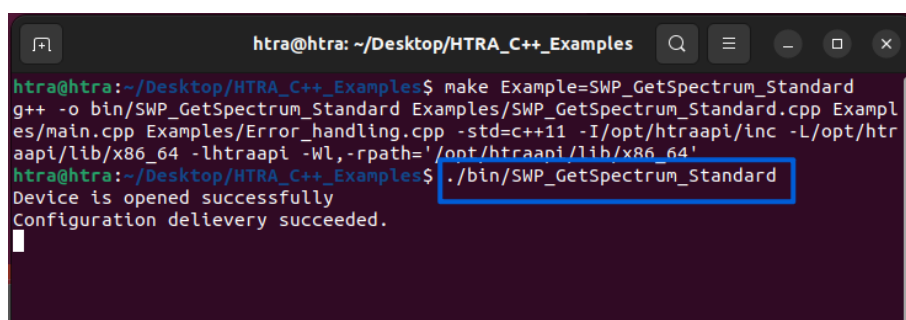


图 44 运行编译完成的 SWP_GetSpectrum_Standard 示例

8.4.2 C++项目创建编译

前提：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，代码编写逻辑符合 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 文件夹用于存放校准文件，创建 htraapi 文件夹用于存放头文件以及动态链接库。

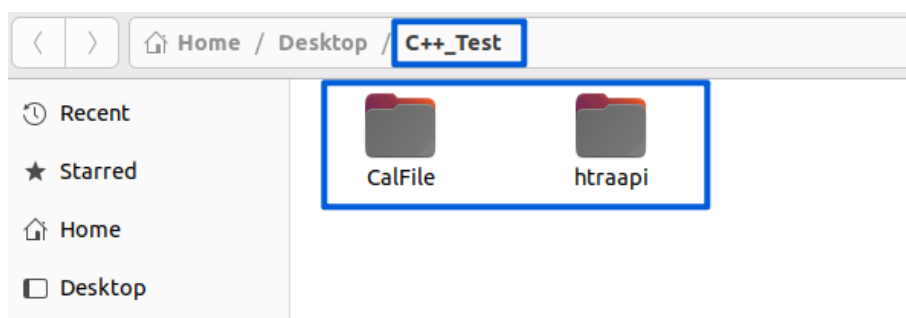


图 45 创建项目文件夹

(2) 在 htraapi 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。

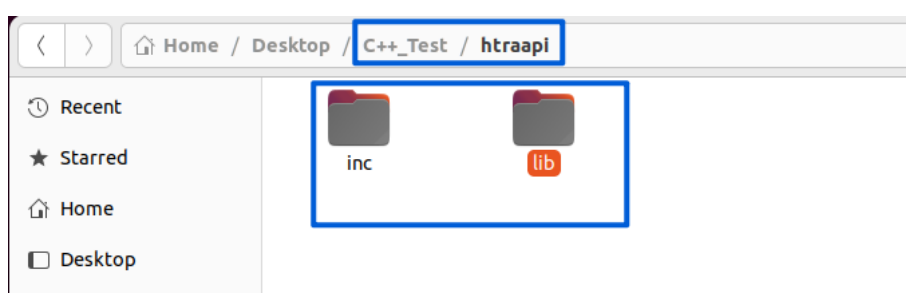


图 46 htraapi 下创建 inc 与 lib 文件夹

(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（SAN-45 R6, SAN-60 R6, SAN-90 R0 设备跳过此步骤）；

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

(5) 参照[环境版本适配性自查章节](#)，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\htraapi\lib”文件夹中。（图示为 x86_64 架构的上位机）；

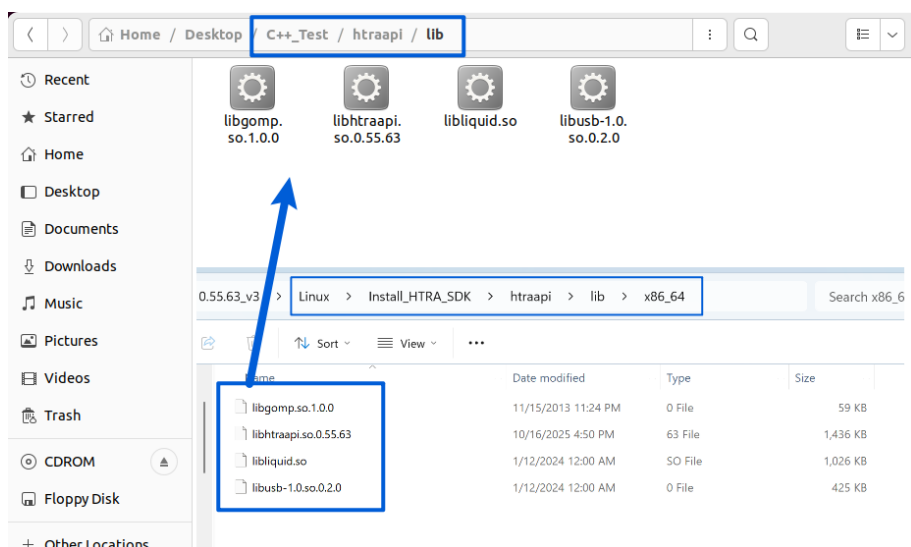


图 47 拷贝对应架构的动态链接库至 lib 文件夹中

(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 libhtraapi.so 库以 0.55.63 版本为例，其他版本修改版本号即可

```
ln -sf libhtraapi.so.0.55.63 libhtraapi.so.0
```

```
ln -sf libhtraapi.so.0 libhtraapi.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
```

```
ln -sf libgomp.so.1.0.0 libgomp.so.1
```

```
ln -sf libgomp.so.1 libgomp.so
```

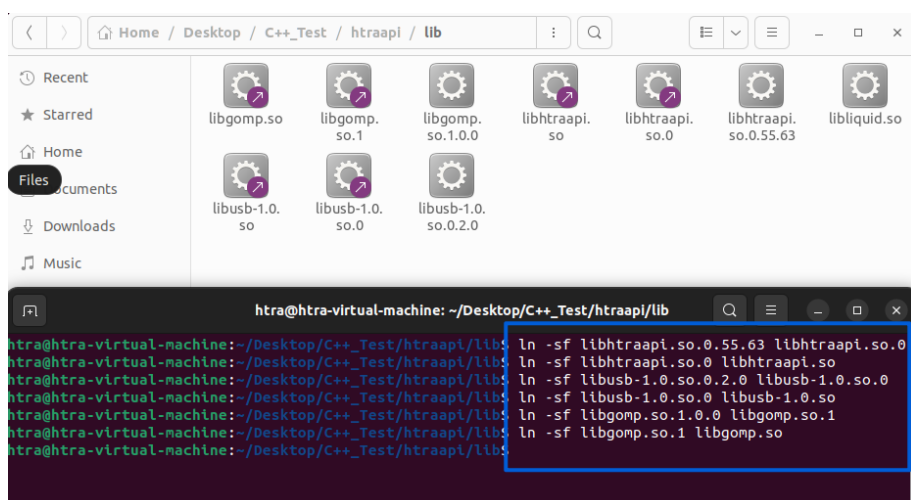


图 48 软链接动态链接库

(7) 将编写好的代码文件存放于 C++_Test 最外层文件夹中;

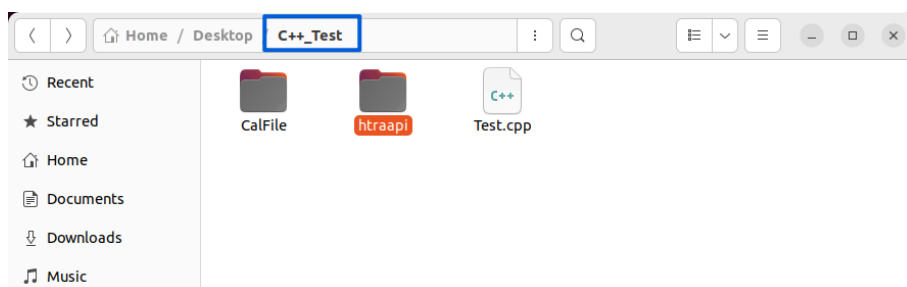


图 49 放置代码文件

(8) 编译生成可执行文件: 根据[环境版本适配性自查章节](#)中确认的系统架构, 在 C++_Test 文件夹终端输入相应架构的命令;

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 系统输入:

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

aarch64 系统输入:

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

armv7 系统输入:

```
arm-linux-gnueabi-g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L ./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'
```

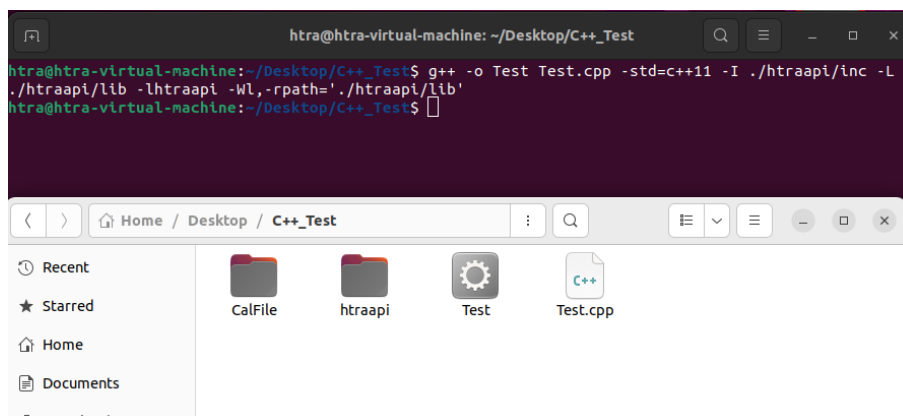
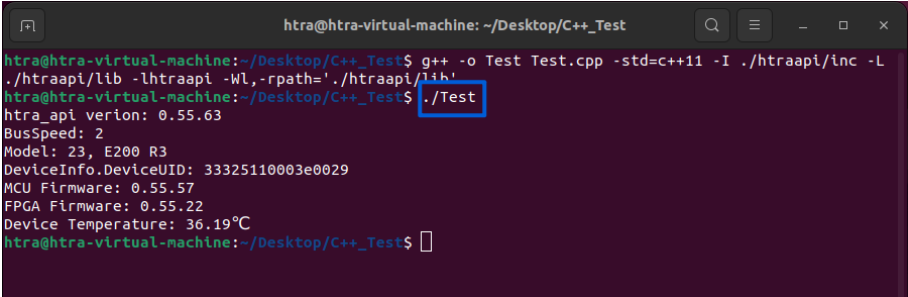


图 50 编译生成可执行文件

(9) 运行程序：输入`./Test` 启动可执行文件。



```
htra@htra-virtual-machine: ~/Desktop/C++_Test
htra@htra-virtual-machine:~/Desktop/C++_Test$ g++ -o Test Test.cpp -std=c++11 -I ./htraapi/inc -L
./htraapi/lib -lhtraapi -Wl,-rpath='./htraapi/lib'
htra@htra-virtual-machine:~/Desktop/C++_Test$ ./Test
htra_api version: 0.55.63
BusSpeed: 2
Model: 23, E200 R3
DeviceInfo.DeviceUID: 33325110003e0029
MCU Firmware: 0.55.57
FPGA Firmware: 0.55.22
Device Temperature: 36.19°C
htra@htra-virtual-machine:~/Desktop/C++_Test$
```

图 51 运行 Test 可执行文件

8.4.3 C++项目交叉编译

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 [C++项目创建编译章节](#) 中步骤 (1) - (8)，创建项目，放置校准文件与头文件，并添加交叉编译目标架构 (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；

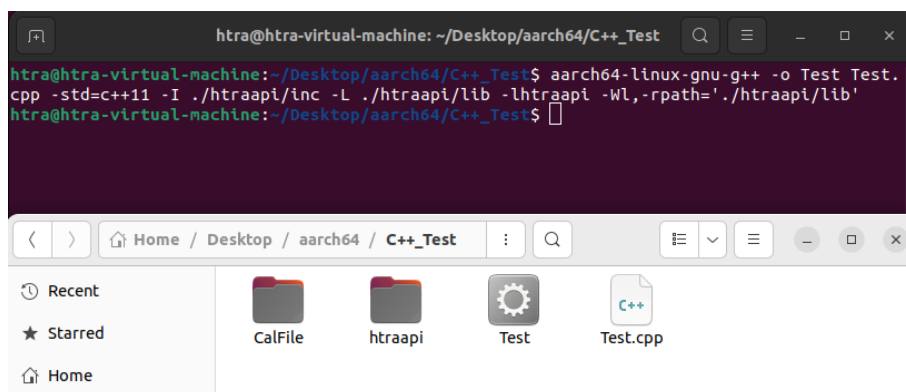


图 52 交叉编译 aarch64 架构的可执行程序

(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；

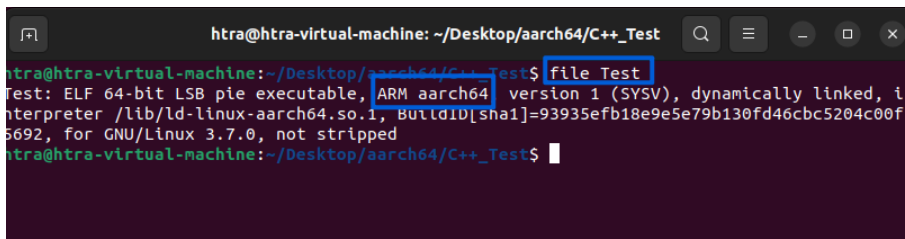


图 53 查看可执行程序架构

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;

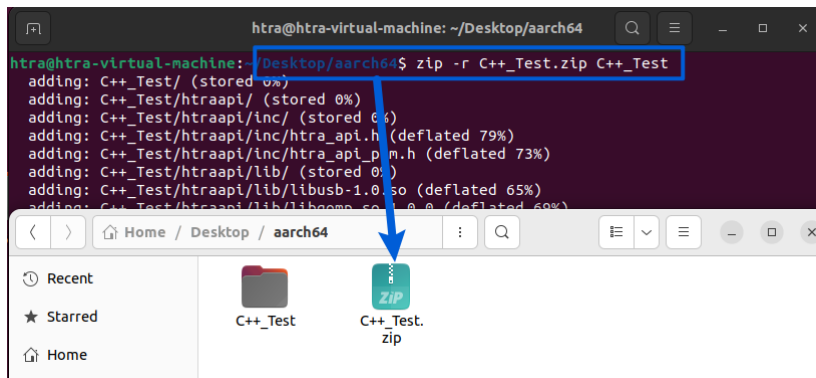


图 54 压缩所创建的项目

- (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。

8.5 Qt 范例使用以及项目创建

8.5.1 Qt 范例使用

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 ubuntu22.04 系统, x86_64 架构下运行 Qt 范例为例。

1. 将随寄 U 盘中“Linux\HTRA_Qt_Examples”文件夹拷贝至上位机;
2. 参照[环境版本适配性自查章节](#)查看系统架构, 并将随寄 U 盘“Linux\Install_HTRA_SDK\htraapi\

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

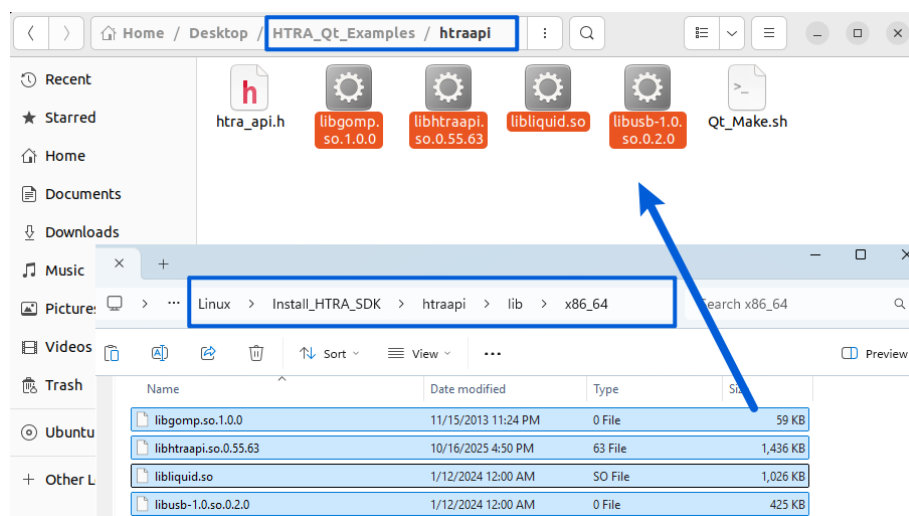


图 55 拷贝对应架构的动态链接库

3. 在 htraapi 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；

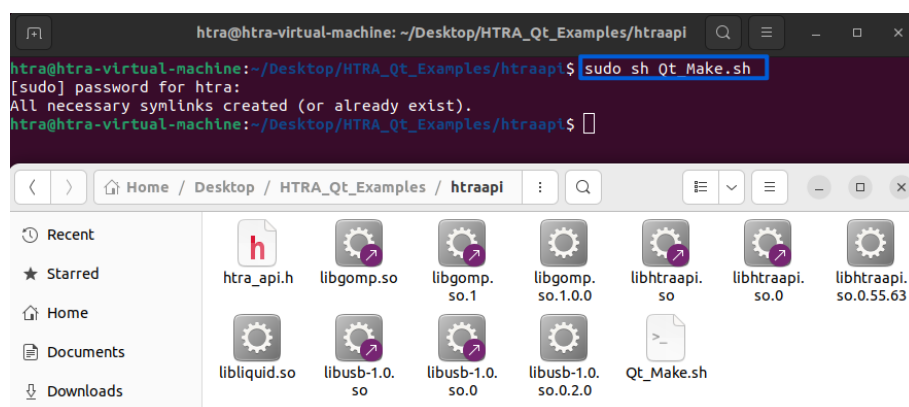


图 56 创建动态链接库的软链接

4. 进入“HTRA_Qt_Examples\bin\CalFile”文件夹，确保文件夹中包含所用设备的校准文件；

5. 使用 QT Creator 打开“HTRA_Qt_Examples\htrademo”中 htrademo.pro 文件，并配置项目的构建环境；

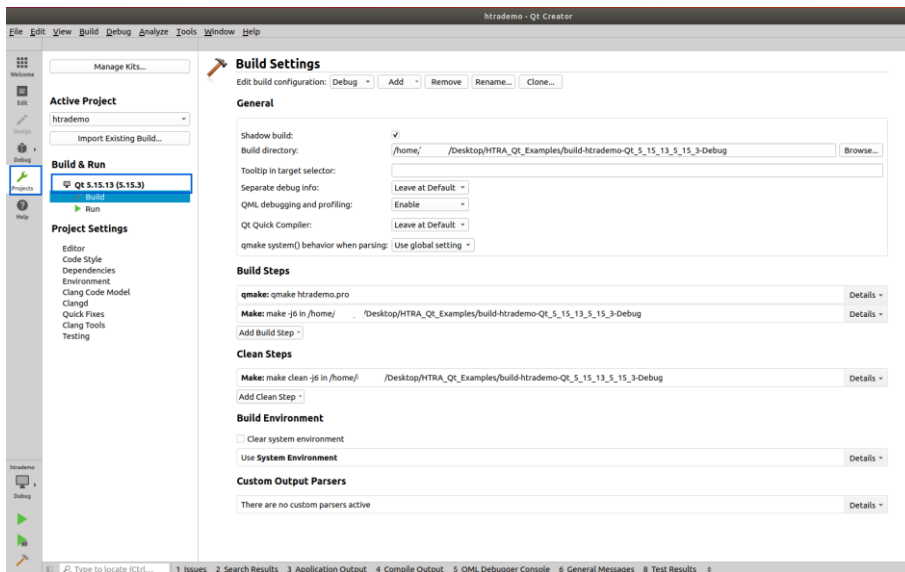


图 57 构建

6. 点击“编辑”，打开“htradio”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。

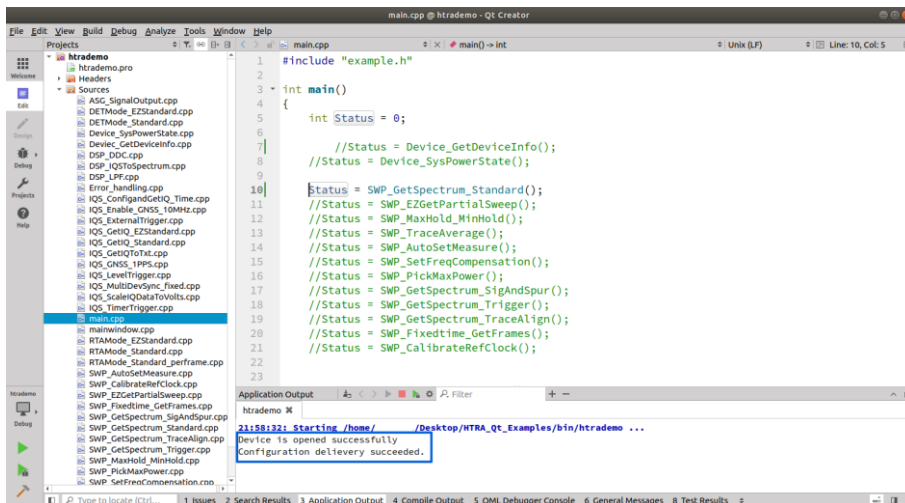


图 58 运行 SWP_GetSpectrum_Standard 示例

8.5.2 Qt 项目创建编译

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 htraapi (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_HTRA_SDK\htraapi\inc”文件夹中的头文件拷贝至“QtTest\htraapi”文件夹中;
4. 将随寄 U 盘中“Linux\Install_HTRA_SDK\htraapi\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\htraapi”文件夹中 (此处以 x86_64 架构的上位机为例);

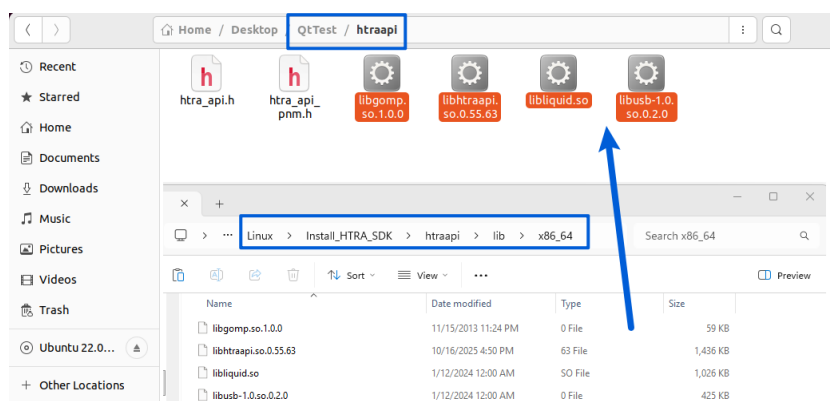


图 59 拷贝对应架构的动态链接库

5. 在 htraapi 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 libhtraapi.so 库以 0.55.63 版本为例, 其他版本修改版本号即可

```
ln -sf libhtraapi.so.0.55.63 libhtraapi.so.0
ln -sf libhtraapi.so.0 libhtraapi.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
ln -sf libgomp.so.1.0.0 libgomp.so.1
ln -sf libgomp.so.1 libgomp.so
```

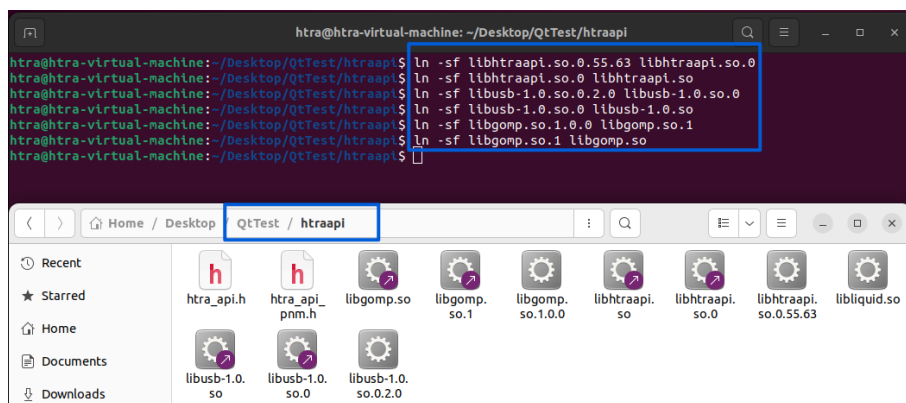


图 60 htraapi 中对动态链接库进行软链接

6. 打开 Qtcreator，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。在此界面中选择一个构建环境，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\htra_api”中的 libhtraapi.so，并点击“打开”，选择“Linux”平台，点击“下一步”。

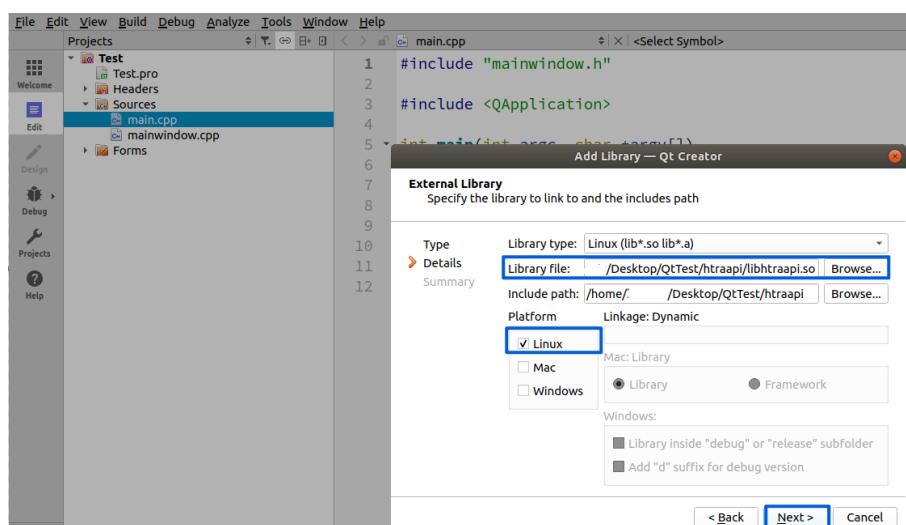


图 61 选择 Linux 平台

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；

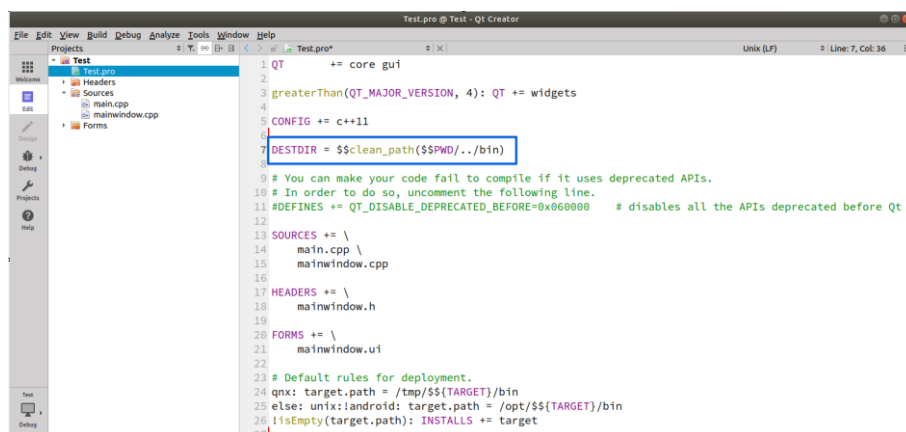


图 62 添加可执行程序生成位置

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

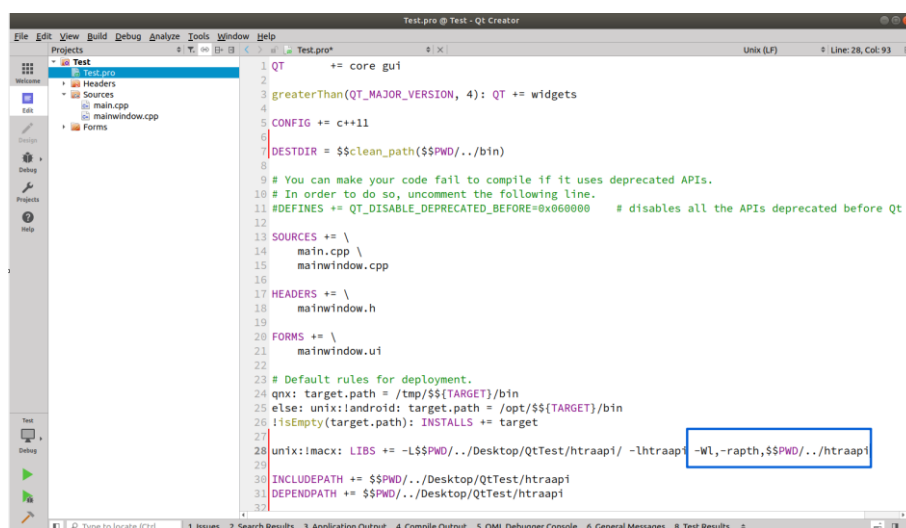


图 63 指定可执行程序的链接库路径

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；

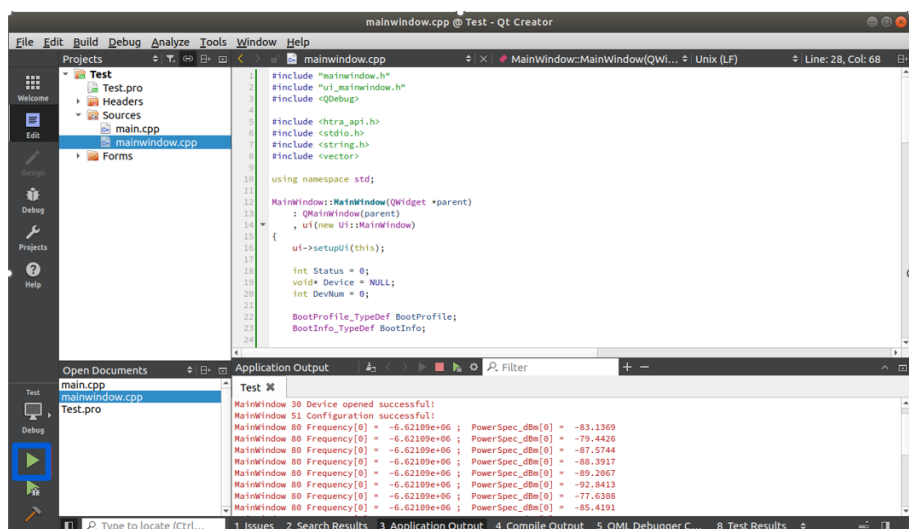


图 64 运行代码

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

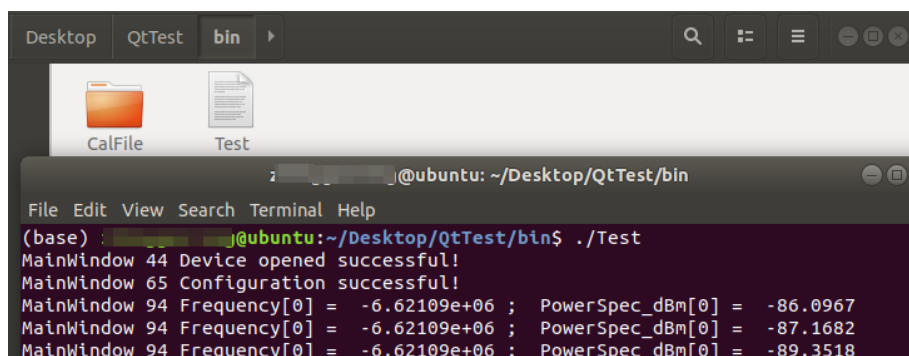


图 65 运行可执行程序

8.5.3 Qt 项目交叉编译

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 arrch64 可执行程序为例）：

1. 生成目标架构可执行文件：

(1) 按照 [Qt 项目创建编译](#) 中窗体程序创建流程的 1-9 步，创建项目并放置校准文件与头文件、放置交叉编译目标架构（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；

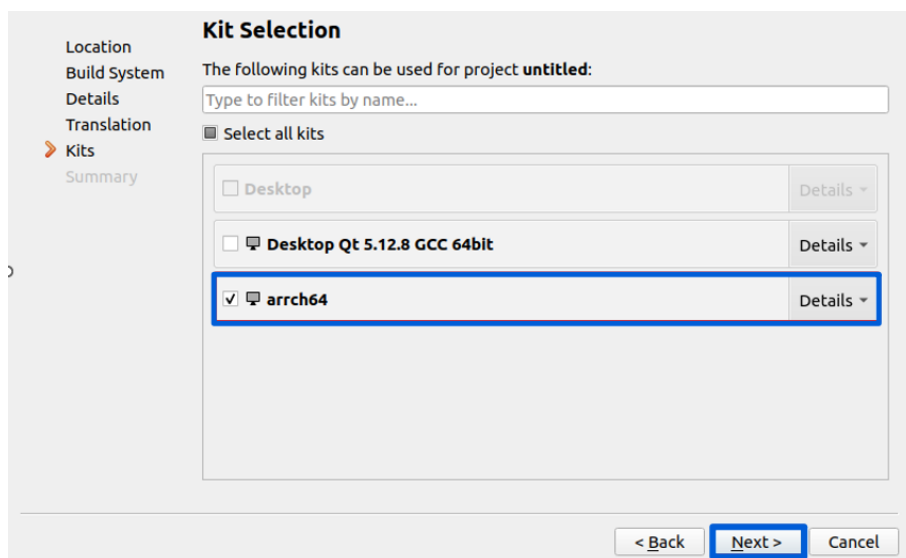


图 66 选择交叉编译目标架构的目标套件

(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;

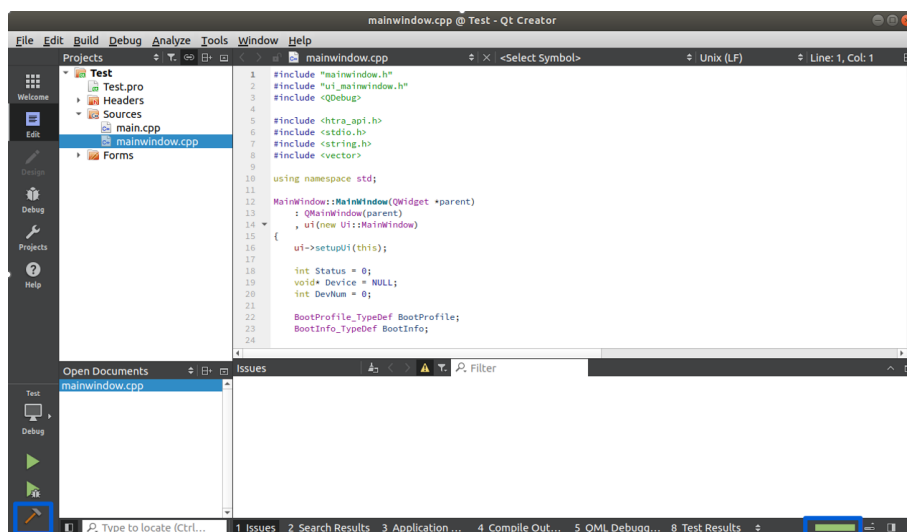


图 67 构建 aarch64 架构的可执行程序

(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)。

```
htra@ubuntu: ~/Desktop/QtTest/bin
File Edit View Search Terminal Help
htra@ubuntu:~/Desktop/QtTest/bin$ file Test
Test: ELF 64-bit LSB shared object, ARM aarch64, version 1 (SYSV), dynamically l
inked, interpreter /lib/ld-linux-aarch64.so.1, for GNU/Linux 3.7.0, BuildID[sha1
]=664b014870dba5dcb7e25e3eb17432ef378b7b1d, not stripped
htra@ubuntu:~/Desktop/QtTest/bin$
```

图 68 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 arrch64 上位机中；

```
htra@ubuntu: ~/Desktop
File Edit View Search Terminal Help
htra@ubuntu:~$ cd Desktop/
htra@ubuntu:~/Desktop$ zip -r QtTest.zip QtTest
updating: QtTest/ (stored 0%)
updating: QtTest/htraapi/ (stored 0%)
updating: QtTest/htraapi/libliquid.so (deflated 60%)
updating: QtTest/htraapi/libhtraapi.so.0.55.53 (deflated 64%)
updating: QtTest/htraapi/libusb-1.0.so.0 (deflated 62%)
updating: QtTest/htraapi/libhtraapi.so.0 (deflated 64%)
updating: QtTest/htraapi/libusb-1.0.so.0.2.0 (deflated 62%)
updating: QtTest/htraapi/libusb-1.0.so (deflated 62%)
updating: QtTest/htraapi/libhtraapi.so (deflated 64%)
updating: QtTest/htraapi/htra_api.h (deflated 79%)
updating: QtTest/Test/ (stored 0%)
updating: QtTest/Test/main.cpp (deflated 27%)
```

图 69 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 arrch64 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。

```
rpdkj@localhost: ~/Desktop/QtTest/bin
File Edit Tabs Help
rpdkj@localhost:~/Desktop/QtTest/bin$ chmod 777 Test
rpdkj@localhost:~/Desktop/QtTest/bin$ ./Test
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '5'.
Failed creating base context during opening of kernel driver.
Kernel module may not have been loaded
arm_release_ver of this libmali is 'g6p0-01eac0', rk_so_ver is '5'.
Failed creating base context during opening of kernel driver.
Kernel module may not have been loaded
MainWindow 44 Device opened successful!
MainWindow 65 Configuration successful!
MainWindow 94 Frequency[0] = -1.2e+06 ; PowerSpec_dBm[0] = -38.5426
MainWindow 94 Frequency[0] = -1.2e+06 ; PowerSpec_dBm[0] = -36.4309
MainWindow 94 Frequency[0] = -1.2e+06 ; PowerSpec_dBm[0] = -32.2113
MainWindow 94 Frequency[0] = -1.2e+06 ; PowerSpec_dBm[0] = -29.7813
MainWindow 94 Frequency[0] = -1.2e+06 ; PowerSpec_dBm[0] = -29.3889
```

图 70 提供权限并运行可执行程序

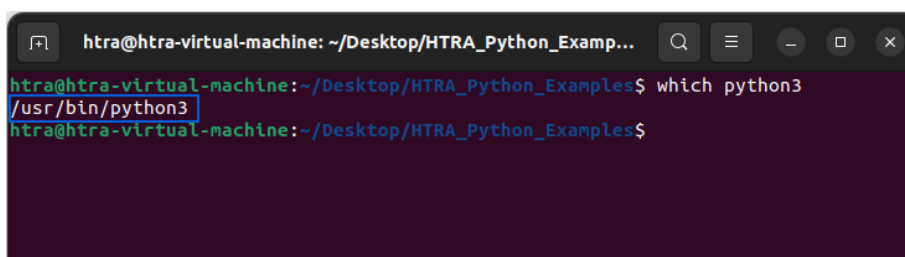
8.6 Python 范例使用以及项目创建

8.6.1 Python 范例使用

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。Python 范例的功能介绍可参考 [Python 范例说明章节](#)。

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

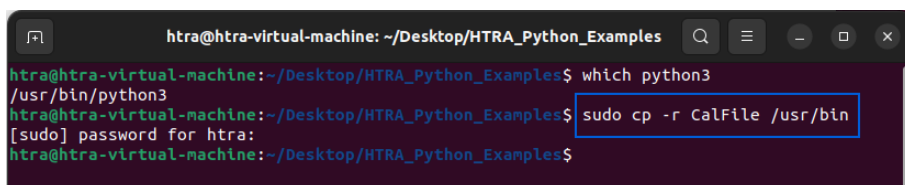
输入“which python3”查看 Python 解释器位置（例如此处位置为/usr/bin）。



```
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples$ which python3
/usr/bin/python3
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples$
```

图 71 查看 Python3 解释器位置

2. 根据获取的解释器地址，输入“sudo cp -r CalFile /usr/bin”，将设备校准文件复制到 Python 解释器同级目录。（如果解释器位置不在/usr/bin，请将路径修改为实际地址）。



```
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples$ which python3
/usr/bin/python3
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples$ sudo cp -r CalFile /usr/bin
[sudo] password for htra:
htra@htra-virtual-machine: ~/Desktop/HTRA_Python_Examples$
```

图 72 拷贝校准文件至解释器同级目录

3. 参照[环境版本适配性自查章节](#)查看系统架构，并将随寄 U 盘“Linux\Install_HTRA_SDK\htraapi\lib”文件夹中，对应架构文件夹中的内容复制至上位机“HTRA_Python_Examples\htraapi”文件夹中；

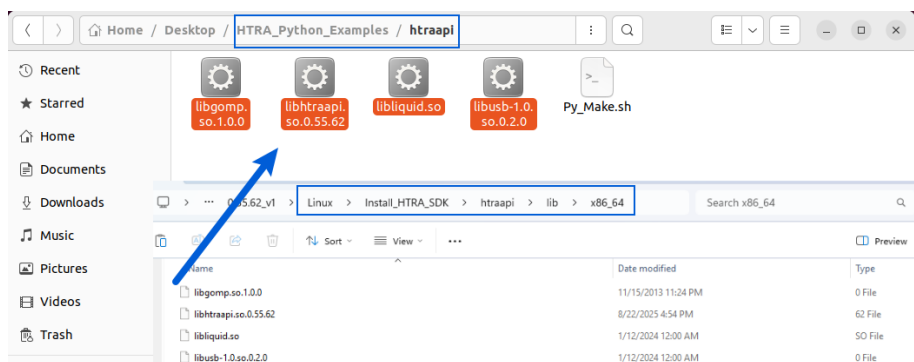


图 73 拷贝动态链接库

4. 在 htraapi 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。

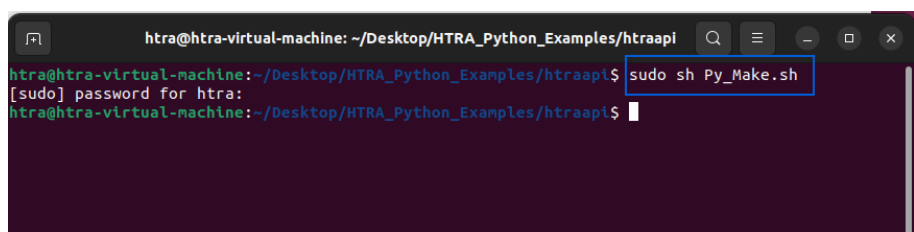


图 74 htraapi 下对库进行软链接

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

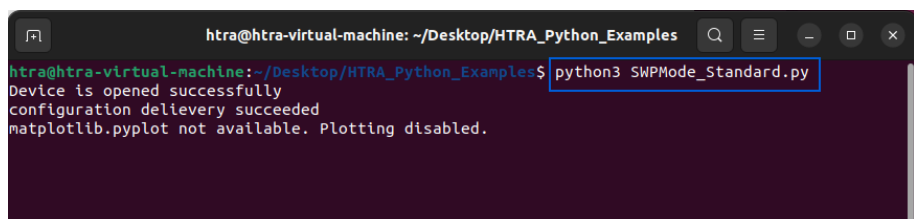


图 75 运行 python 范例

注意: 若使用另一台设备运行 python 范例, 请先将校准文件拷贝到“HTRA_Python_Examples\CalFile”文件夹下, 然后在 HTRA_Python_Example 文件夹下的终端输入“sudo cp -r CalFile /usr/bin”, 更新校准文件至 Python 解释器同级目录。

8.6.2 Python 项目创建

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（HTRA_Python_Example）下，按照[Python 范例使用](#)章节流程执行即可。

 www.harogic.cn

 cninfo@harogic.com

 +025-8330 5049