ICX Series API

Examples Usage Guide

Berkeley Nucleonics Corporation
www.berkeleynucleonics.com

Contents

1. Version Management

Updated Description Sheet

Version	Description	Date
V2.1	1. Added: detector data recording and real-time spectrum data recording examples in C/C++ 2. Modified: compilation environment requirements	08/05/2026
V2.0	3. Initial Version	12/05/2025

2. C/C++

2.1 Configure Development Environment

The following uses configuring Debug Win32 as an example. For other build types and platforms, please refer to the end of this chapter.

1. Open Visual Studio 2019 and create a new project;
2. After creation is completed, copy the "Windows\HTRA_API" folder from the delivery USB drive to a directory at the same level as the project;

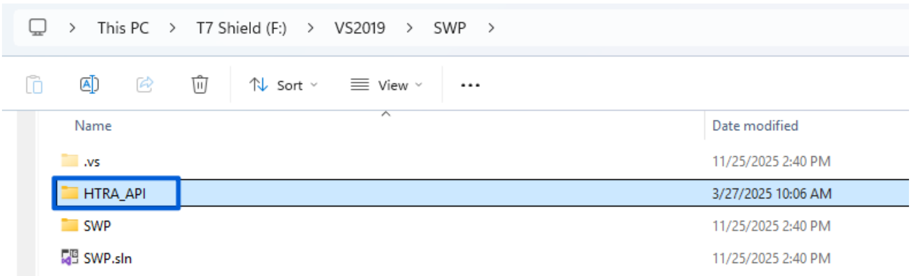

Figure 1 Copy the HTRA_API file

3. Double-click to open SWP.sln, right-click on "Source Files" -> "Add" -> "New Item" -> "C++ File (.cpp)" to create a new SWP.cpp file;
4. Click "Project" -> "Properties" in the menu bar, set "Configuration" to "Debug" and "Platform" to "Win32";
5. In Configuration Properties -> Debugging, set the Environment variable to "Path=..\HTRA_API\x86";

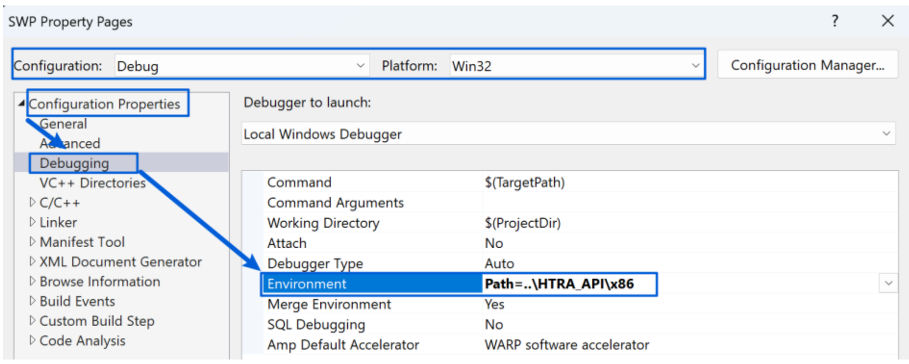

Figure 2 Configure environment variables

6. In Configuration Properties -> C/C++ -> General, set the Additional Include Directories to "$(SolutionDir)\HTRA_API\x86";

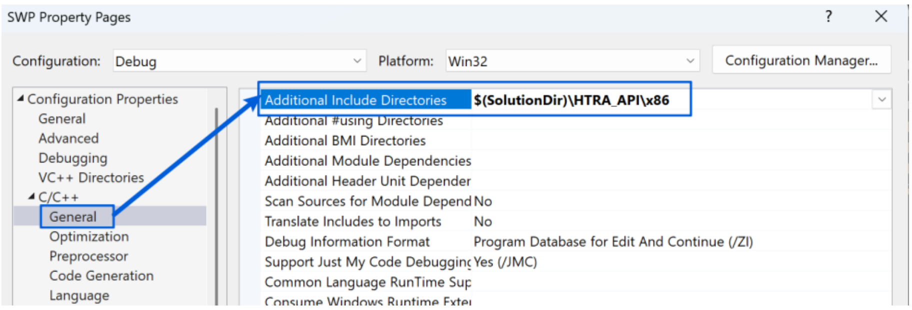

Figure 3 Add additional include directories

7. Set the "Additional Library Directories" under Configuration Properties -> Linker -> General to "$(SolutionDir)\HTRA_API\x86";

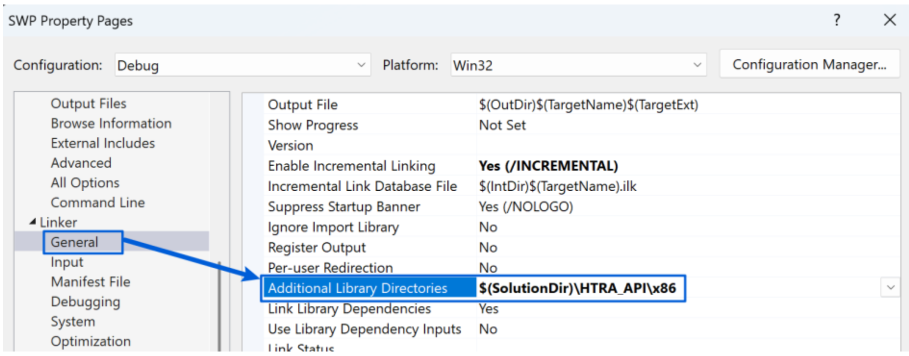

Figure 4 Configure additional library directories

8. Add "htra_api.lib" to "Additional Dependencies" under Configuration Properties -> Linker -> Input, then click "OK".

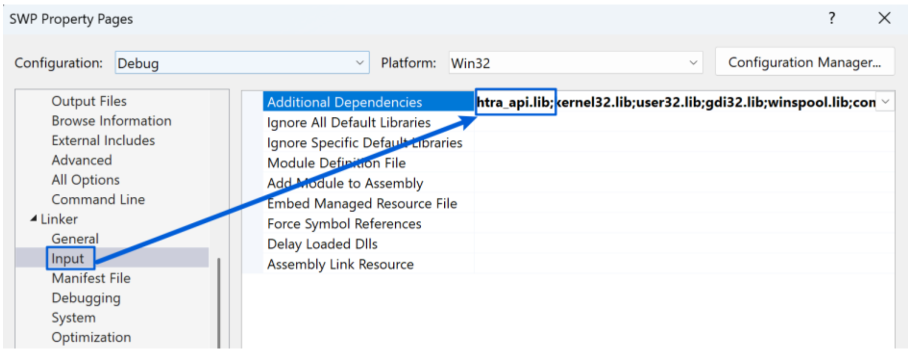

Figure 5 Configure additional dependencies

9. At this point, the environment configuration is complete. You may now refer to the examples provided in the "Windows\HTRA_H1P_API_Example\HTRA_H1P_C++_Examples" directory on the shipped USB flash drive for application development.

- Configure Release Win32: Set "Configuration" in Step 4 to "Release", and keep all other settings unchanged;
- Configure Debug x64: Set "Configuration" in Step 4 to "Debug" and "Platform" to "x64". Also replace all environment variables or additional directory paths containing "HTRA_API\x86" with "HTRA_API\x64";
- Configure Release x64: Set "Configuration" in Step 4 to "Release" and "Platform" to "x64".

Also replace all environment variables or additional directory paths containing "HTRA_API\x86" with "HTRA_API\x64".

2.2 Usage Process for C++ Examples

Note: Only one example can run at a time, and the example cannot run simultaneously with SAStudio4.

2.2.1 Usage of General C++ Example

1. Use Visual Studio to open the solution HTRA_C++_Examples.sln located in the folder "Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\HTRA_C++_Examples" on the provided USB drive;
2. Click the HTRA_C++_Examples project on the right, then open the main.cpp file under the Source Files folder;
3. Each example program is encapsulated in a separate function. To use one, uncomment the corresponding routine, save the file, select the build architecture, and then click Run.

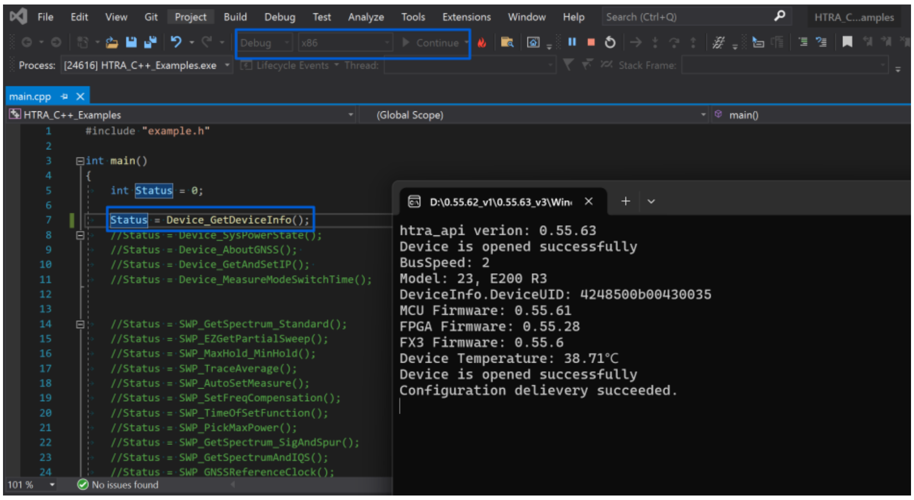

Figure 6 Run the Device_GetDeviceInfo example

2.2.2 Usage of the AM/FM Demodulation Example

1. Open the solution Htra_Demodulation.sln in "Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\Htra_Demodulation" on the supplied USB drive using Visual Studio;
2. Click the Htra_Demodulation project on the right, then open the main.cpp file under the Source Files folder;
3. In the C++ example package, each demo routine is encapsulated in a separate function. To run a demo, uncomment the corresponding routine, save the file, select the build architecture, and click Run.

2.2.3 Usage of the Recording and Playback Example

1. Use Visual Studio to open the solution Htra_RecordingandPlayBack.sln located in the folder "Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\Htra_RecordingandPlayBack" on the provided USB drive;
2. Double-click the Htra_RecordingandPlayBack project on the right to open the main.cpp file in the source files;
3. Each routine in the recording and playback example is encapsulated in a separate function. To use the example, simply uncomment it.

- Usage of the Reading Example

1) For example, when reading IQS mode stream disk data, uncomment the IQSMode_PlayBack function and save;
2) Place the recording file generated in IQS mode into the folder "Windows\HTRA_H1P_API_Example\HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data";
3) Click to enter IQSMode_PlayBack.cpp and modify the name of the recorded file in the IQSMode_PlayBack() function;

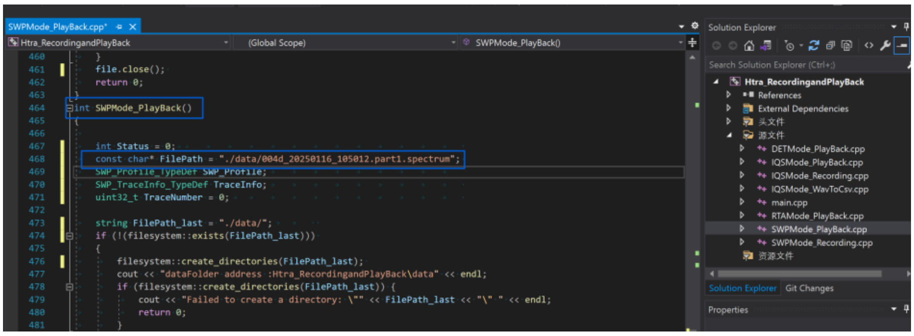

Figure 7 Modifying the recording file name

4) Running the program will generate the parsed data file "IQSMode_Data.txt" in the data folder under the SWP mode.

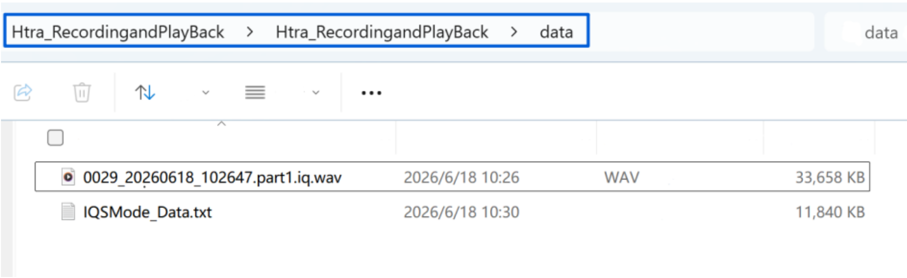

Figure 8 View the parsed data in IQS mode

- Usage of the Recording Example

1) For example, when testing the IQSMode_Recording routine, uncomment and save;

2) Click to enter the IQSMode_Recording() function, configure the relevant parameters, and then run the program. During execution, the file will be continuously recorded until manually stopped;

3) The recording file data in IQS mode can be found in the "HTRA_C++_Examples\Htra_RecordingandPlayBack\Htra_RecordingandPlayBack\data" folder.

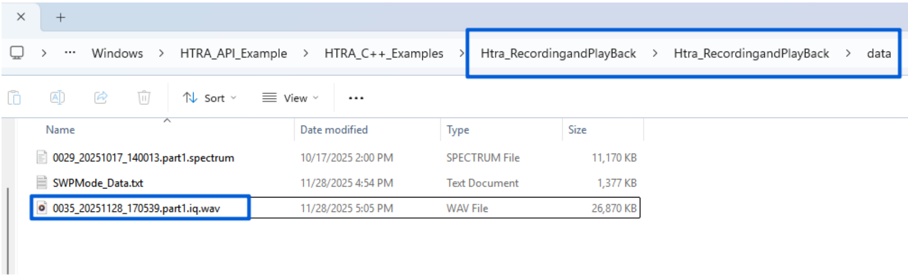

Figure 9 View the recorded file data in IQ streaming mode

2.3 Device-related

2.3.1 Get Device Information

Device_GetDeviceInfo.cpp: Get device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

2.3.2 Device Standby

Device_SysPowerState.cpp: An example of setting the device's standby state, which can be configured to normal operating state or RF powered down state (low power).

2.3.3 GNSS-related

Device_AboutGNSS.cpp: Retrieves information such as latitude, longitude, altitude, and time obtained from the GNSS module, acquires GNSS-related latitude and time information from MeasAuxInfo in SWP mode, and obtains latitude and time information from IQStream.DeviceState in IQS mode.

2.3.4 Get and modify the IP address of the NX device

Device_GetAndSetIP.cpp: Retrieve the device's IP address and modify it using the device UID or the device's current IP address.

2.3.5 Mode switching time consumption

Device_MeasureModeSwitchTime.cpp: Get the time required for the current host computer to switch between different modes.

2.4 SWP Mode

2.4.1 Get standard spectrum

SWP_GetSpectrum_Standard.cpp: Obtain spectrum data by calling the function interface.

2.4.2 Maximum and minimum hold

SWP_MaxHold_MinHold.cpp: Set the trace mode to MaxHold or MinHold, and use SWP_ResetTraceHold to reset the hold.

2.4.3 Average Trace

SWP_TraceAverage.cpp: Average processing of the acquired trace.

2.4.4 Automatic Configuration Measurement

SWP_AutoSetMeasure.cpp: Automatically configures relevant parameters based on specific SWP applications, completing measurements by issuing automatic configuration parameters.

2.4.5 Frequency Compensation

SWP_SetFreqCompensation.cpp: When an external attenuator is present, compensation can be applied to the corresponding frequency band to ensure that the test results remain accurate.

2.4.6 Function execution time, sweep speed, and throughput

SWP_TimeOfSetFunction.cpp: Obtains the call duration of the SWP_Configuration, SWP_GetPartialSweep, and SWP_GetFullSweep functions, along with the sweep speed and throughput under the current configuration.

2.4.7 Obtain spectrum peak values

SWP_PickMaxPower.cpp: Obtain the maximum power point of the current spectrum and its corresponding frequency point.

2.4.8 Signals and Spurious

SWP_GetSpectrum_SigAndSpur.cpp: This can distinguish between signals and spurious after obtaining spectrum data.

2.4.9 Simultaneous Acquisition of Spectrum and IQ

SWP_GetSpectrumAndIQS.cpp: This allows for the simultaneous acquisition of spectrum data and IQ data.

2.4.10 Using GNSS 10MHz Reference Clock

SWP_GNSSReferenceClock.cpp: This uses a high-quality GNSS module's 10MHz reference clock in SWP mode.

2.4.11 External Trigger Mode

SWP_GetSpectrum_Trigger.cpp: This obtains spectrum data when the trigger source is set to external trigger.

2.4.12 Trace Alignment Method

SWP_GetSpectrum_TraceAlign.cpp: Obtain spectrum data when the trace alignment method is set to align to the starting frequency or align to the center frequency.

2.4.13 Number of Spectrum Frames Obtainable within a Certain Time

SWP_Fixedtime_GetFrames.cpp: Loop 50 times to obtain 10 seconds of spectrum data, resulting in the average number of spectrum frames that can be obtained within 10 seconds.

2.4.14 External Trigger Calibration of Internal 10MHz Reference Clock

SWP_GetSpectrum_Calibration.cpp: An example of the device calibrating the clock via GNSS-1PPS or through external trigger.

2.4.15 Phase noise measurement

SWP_Meas_PhaseNoise.cpp: Measures the single-sideband phase noise (unit: dBc/Hz) of the received signal at the user-specified frequency deviation, supports multi-frequency point configuration.

2.4.16 Channel power measurement

SWP_Meas_ChannelPower.cpp: Measures the channel power of the received signal.

2.4.17 Adjacent Channel Power Ratio Measurement

SWP_Meas_ACPR.cpp: Measure the adjacent channel power ratio of the received signal.

2.4.18 Percentage occupied bandwidth Measurement

SWP_Meas_OBW.cpp: Measures the occupied bandwidth of the received signal in percentage.

2.4.19 XdB Occupied Bandwidth Measurement

SWP_Meas_XdBBW.cpp: Measures the occupied bandwidth of the received signal in XdB.

2.4.20 IM3 measurements

SWP_Meas_IM3.cpp: IM3 measurement of the received signal.

2.4.21 Frequency interval matching RBW

SWP_RBW_Spaced_Trace.cpp: Makes the frequency interval between two points of the trace close to the set RBW value.

2.4.22 Using an External 10MHz Reference Clock

SWP_RefCLKSource_External.cpp: Use external 10MHz reference clock. (Take the use of external 10MHz reference clock in SWP mode as an example, the same method is used in other modes).

2.4.23 Spectrum Data Recording

SWPMode_Recording.cpp: Records the spectrum data acquired in SWP mode and generates a .spectrum format file.

2.4.24 Spectrum Data Reading

SWPMode_PlayBack.cpp: Reads the .spectrum file recorded in SWP mode and parses the data into a .txt file.

2.5 IQS Mode

2.5.1 Obtain Fixed Number or Continuous Stream of IQ Data

IQS_GetIQ_Standard.cpp: Obtain a fixed number or continuous stream of IQ data under professional configuration.

2.5.2 IQ data converted to voltage V units

IQS_ScaleIQDataToVolts.cpp: Converts the acquired IQ data into data in volts (V).

2.5.3 Time taken to issue configuration and acquire IQ

IQS_ConfigandGetIQ_Time.cpp: Obtains the call duration of the IQS_Configuration and IQS_GetIQStream_PM1 functions.

2.5.4 IQ to Spectrum Data

DSP_IQSToSpectrum.cpp: Converts the acquired time-domain IQ data into spectrum data using spectral analysis methods.

2.5.5 IQ to Spectrum (using liquid library version)

IQS_ToSpectrumByLiquid.cpp: Uses the liquid library to convert the acquired time-domain IQ data into spectrum data through spectral analysis methods.

2.5.6 FM Demodulation Playback

DSP_FMDemod.cpp: Performs FM demodulation on the IQ data and plays the demodulated audio.

2.5.7 FM Demodulation Data Analysis

IQS_FMDataAnalysis.cpp: Audio analysis of the IQ data after FM demodulation to get the audio voltage, audio frequency, SINAD and total harmonic distortion.

2.5.8 AM Demodulation Playback

DSP_AM_Demod.cpp: Performs AM demodulation on IQ data and plays the demodulated audio.

2.5.9 AM Demodulation Data Analysis

IQS_FMDataAnalysis.cpp: Audio analysis of the IQ data after AM demodulation to get the audio voltage, audio frequency, SINAD and total harmonic distortion.

2.5.10 Digital Downconversion

DSP_DDC.cpp: Resamples the obtained IQ data.

2.5.11 Digital Low-Pass Filter

DSP_LPF.cpp: Perform low-pass filtering on the obtained IQ data.

2.5.12 Audio Analysis

IQS_AudioAnalysis.cpp: Performs audio analysis on the demodulated IQ data to obtain audio voltage, audio frequency, SINAD, and total harmonic distortion.

2.5.13 Read the IQS stream disk data from Application Software

IQSMode_PlayBack.cpp: Parses the recorded file data in IQS mode from Application Software and writes the read spectrum data into the IQSMode_Data.txt file.

2.5.14 Record IQ data in .wav format

IQSMode_Recording.cpp: Stores the acquired IQ data in .wav format.

2.5.15 .wav changed to .csv

IQSMode_WavToCsv.cpp: Parses and extracts I and Q channel data from the .wav recording file data in IQS mode, converting it into a .CSV format file for saving.

2.5.16 Streaming and reading IQ data

IQS_GetIQToTxt.cpp: Writes the obtained IQ data into a .txt file.

2.5.17 Multithreaded acquisition, processing, and streaming of IQ data

IQS_Multithread_GetIQ_FFT_Write: Simultaneously acquires IQ data, performs FFT, and writes IQ data into a .txt file.

2.5.18 GNSS1PPS trigger

IQS_GNSS_1PPS.cpp: Configure the trigger source to be the 1PPS signal provided by the internal GNSS system.

2.5.19 IQS multi-device synchronization

IQS_MultiDevSync_fixed.cpp: Multiple devices simultaneously collect the same signal at the same time.

2.5.20 External Trigger

IQS_ExternalTrigger.cpp: Configure the trigger source as external trigger.

2.5.21 Timer Trigger

IQS_TimerTrigger.cpp: Configure the trigger source as timer trigger.

2.5.22 Level Trigger(pre-trigger)

IQS_LevelTrigger_PreTriggerr.cpp: Configure the trigger source as level trigger and set the pre-trigger time.

2.5.23 Level Trigger (Trigger delay)

IQS_LevelTrigger_TriggerDelay.cpp: Configure the trigger source to be level trigger and set the trigger delay time.

2.5.24 Using GNSS 10MHz Reference Clock

IQS_Enable_GNSS_10MHz.cpp: Use a high-quality GNSS module's 10MHz reference clock in IQS mode.

2.6 DET Mode

2.6.1 Obtain detection data for fixed points or continuous streams

DETMode_Standard.cpp: Obtain detection data for fixed points or continuous streams.

2.6.2 Detector Data Recording

DETMode_Recording.cpp: Records the power stream data acquired in DET mode and generates a .det.wav format file.

2.6.3 Detector Data Reading

DETMode_PlayBack.cpp: Reads the .det.wav file recorded in DET mode and parses the data into a .txt file.

2.7 RTA Mode

2.7.1 Obtain real-time spectrum data for fixed points or continuous stream

RTAMode_Standard.cpp: Obtain real-time spectrum data for a fixed number of points (duration) or continuous stream.

2.7.2 Time consumption of each frame of data in RTA mode

RTAMode_Standard_perframe.cpp: Acquire 100 frames of data and calculate the average processing time for each frame.

2.7.3 Real-Time Spectrum Data Recording

RTAMode_Recording.cpp: Records the real-time spectrum data acquired in RTA mode and generates a .rtspectrum format file.

2.7.4 Real-Time Spectrum Data Reading

RTAMode_PlayBack.cpp: Reads the .rtspectrum file recorded in RTA mode and parses the data into a .txt file.

2.8 ASG Signal Source (optional)

2.8.1 Output single tone/sweep/power scan signals

ASG_SignalOutput.cpp: Output single tone/sweep/power scan signals as needed.

3. Qt

3.1 Configure Development Environment

The following uses the configuration of an x64 architecture development environment as an example.

1. Create a new folder (taking "QtTest" as an example) to store the project files;
2. Copy the "Windows\HTRA_H1P_API\x64\htra_api" folder from the provided USB drive into the newly created "QtTest" folder;
3. Open Qt Creator, and click "File" -> "New File or Project";
4. In project dialog, click "Application (Qt)", select "Qt Widgets Application", then click "Choose…";
5. Enter the project name (e.g., "test"), click the "Browse" button, select the previously created "QtTest" folder, and then click "Next";
6. Select "qmake", continue by clicking "Next" until the "Kit Selection" screen appears. On this screen, choose a build environment, and then click "Next";

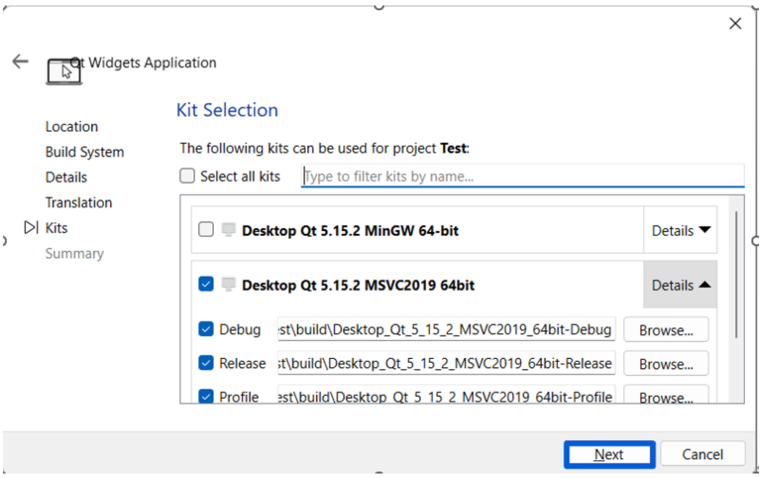

Figure 10 Select the build environment

7. Click "Finish" to create the project;
8. In the Qt Creator main interface, right-click the "Test" project, select "Add Library…" -> "External Library" -> "Next";
9. Click to browse the library file, select the htra_api.lib library in "QtTest\htra_api", and click "Open";
10. Uncheck all options under Windows, then click "Static Library", finally select the Windows platform, and click "Next";

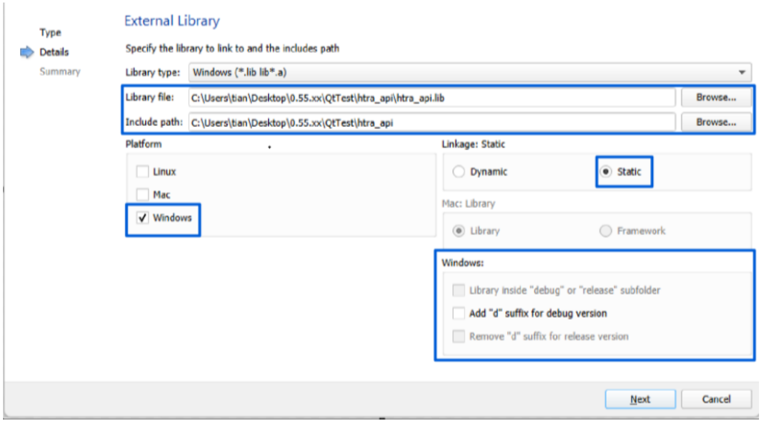

Figure 11 Set the build configuration

11. Click "Finish" to add the external library; Delete the last line *else:win32-g++:PRE_TARGETDEPS+=$$PWD/../htra_api/libhtra_api.a* in the "test.pro" file. After saving, you can write project code normally;

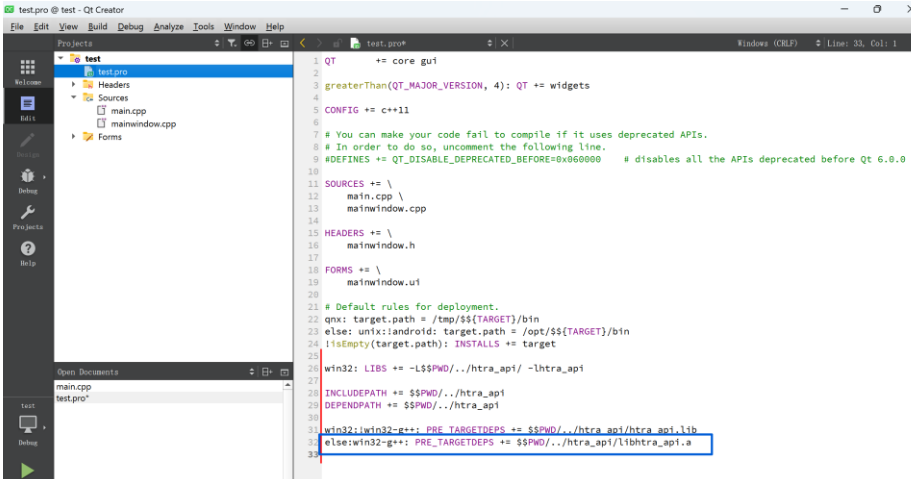

Figure 12 Modify the test.pro file

12. After writing the code, click "Run". The program running correctly is illustrated as follows.

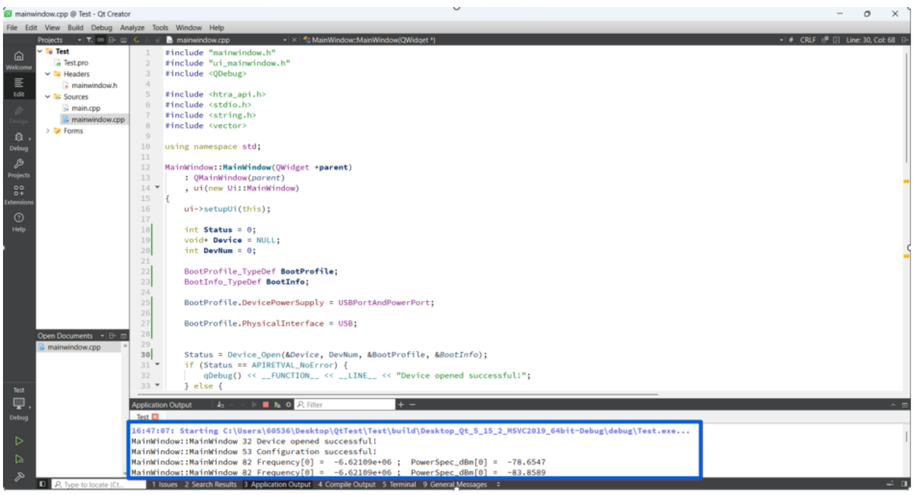

Figure 13 Result

3.2 Qt Example Usage Process

Note: Only one example can run at a time; the example and SAStudio4 cannot run simultaneously.

All Qt examples on the provided USB drive follow the same usage process. The following introduces the usage of the HTRA_Qt_Example project as an example:

1. Use Qt Creator to open the htrademo.pro file located in the "Windows\HTRA_H1P_API_Example\HTRA_Qt_Examples\HTRA_Qt_Example\HTRA_Qt_Example" folder on the USB drive;
2. Click "Projects" to configure a 64-bit build environment for the project (the provided example uses 64-bit library files. If you need to use a 32-bit build environment, replace the libraries in the "Windows\HTRA_H1P_API\x86" folder with the 32-bit libraries from "\Windows\HTRA_API\x86");

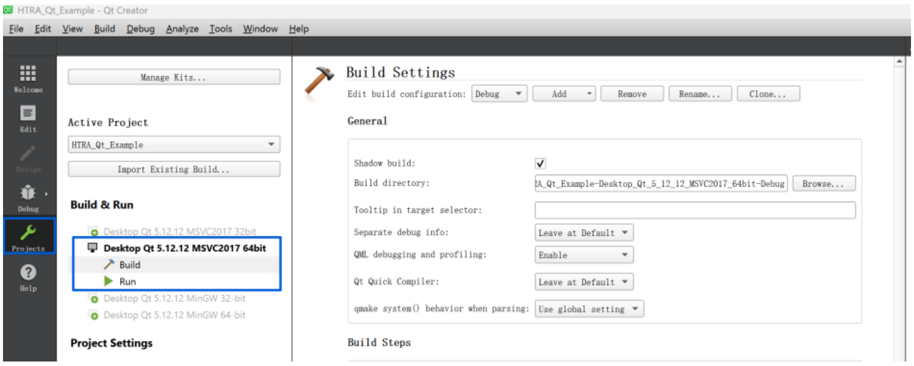

Figure 14 Configure a 64-bit build environment

3. After configuring the environment, click "Edit" to open main.cpp in the Sources folder of the "HTRA_Qt_Example" project. Uncomment the relevant functions, save the file, and click "Run" to execute different examples.

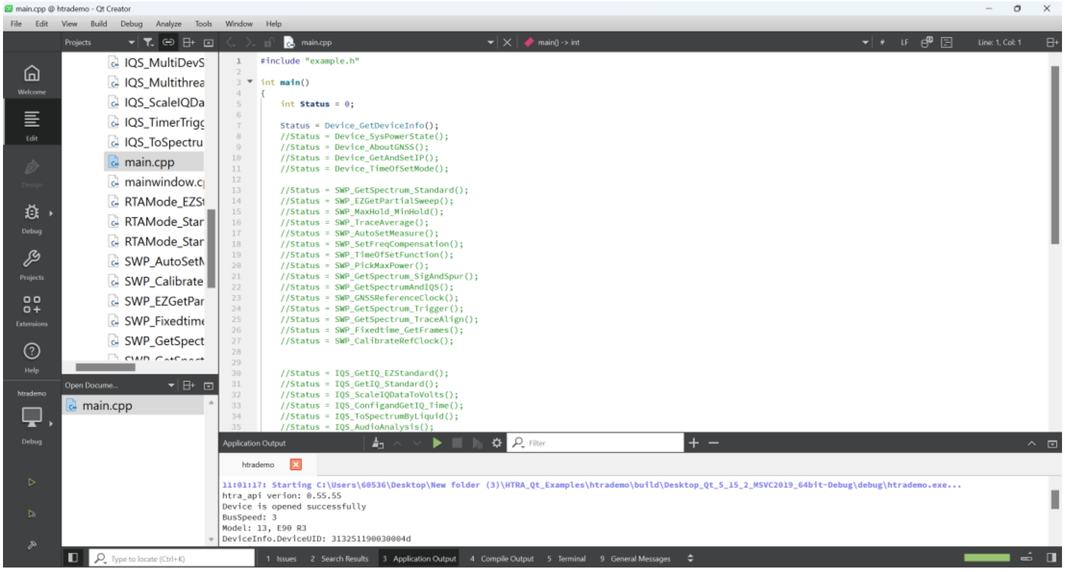

Figure 15 Running the QT Example

3.3 Qt Example Description

The Qt examples on the USB stick include a general example (HTRA_Qt_Example), an FM and AM demodulation example (HTRA_Demodulation), and an IQ to Spectrum example (IQS_ToSpectrumBuLiquid) using the Liquid library, refer to the C/C++ examples chapter for specific examples.

4. Python

4.1 Configure Development Environment

1. Create a new folder on the desktop (e.g., TEST);
2. Open the provided USB drive, go to "\HTRA_H1P_API_Example\HTRA_Python_Examples ", and copy the "HTRA_API" folder and "htra_api.py" file into the newly created test folder;
3. Open Visual Studio Code, click "File" → "Open Folder", and select the newly created test folder;
4. In the left Explorer panel, click the "New File" icon to create a new Python file (e.g., test.py), and you can start writing code normally.

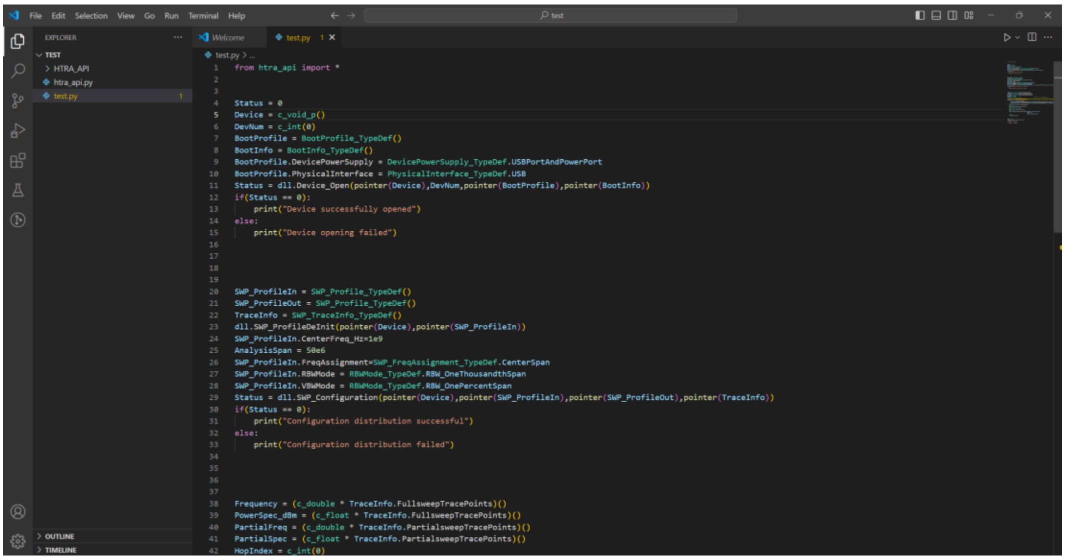

Figure 16 Creating a New Python Project

4.2 Python Example Usage Process

1. Open the "\Windows\HTRA_H1P_API_Example\HTRA_Python_Examples " project folder on the provided USB drive using Visual Studio Code or another IDE. The "htra_api.py" file is the Python mapping for the dynamic library, and the other files are example programs;
2. After properly configuring the Python environment, select any example program (e.g., SWPMode_Standard.py) and run the file directly. The program running successfully is illustrated as follows:

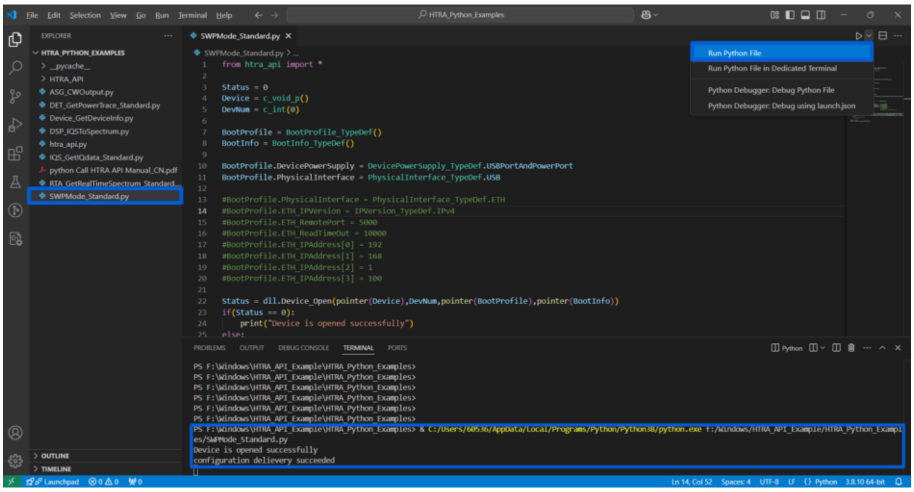

Figure 17 Running SWPMode_Standard.py

4.3 Python Example Description

4.3.1 Get device information

Device_GetDeviceInfo.py: Retrieves various device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

4.3.2 Obtain Standard Spectrum Data

SWPMode_Standard.py: Obtains complete spectrum data within a specified frequency band(Plotting the spectrum if the host computer includes the matplotlib library).

4.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQdata_Standard.py: Obtains IQ data under different trigger modes in IQS mode.

4.3.4 Obtain Power Detection Data for a Fixed Number of Points or Duration

DET_GetPowerTrace_Standard.py: Obtains power detection data under different trigger modes in DET mode(Plotting the Time-power graph if the host computer includes the matplotlib library).

4.3.5 Obtain real-time spectrum data for a fixed number of points or duration

RTA_GetRealTimeSpectrum_Standard.py: Obtain real-time spectrum data under different trigger modes in RTA mode.

4.3.6 IQ to Spectrum Data

DSP_IQSToSpectrum.py: Convert the IQ data obtained in IQS mode into spectrum data(If the host computer contains the matplotlib library then plot the spectrum and IQ time-domain graphs).

4.3.7 GNSS Related

Device_AboutGNSS.py: Obtain information such as longitude, latitude, altitude, and time acquired by the GNSS module; obtain the longitude, latitude, and time information from the MeasAuxInfo structure in IQS mode.

4.3.8 Digital Down Conversion (DDC)

DSP_DDC.py : Resamples the acquired IQ data and plots the corresponding spectrum.

4.3.9 Single-Tone / Frequency Sweep / Power Sweep Signal Output

ASG_SignalOutput.cpp: Outputs single-tone, frequency sweep, or power sweep signals as required.

5. Matlab

The following uses Matlab 2016a as an example to demonstrate how to call the 64-bit htra_api dynamic link library.

5.1 Running the Matlab Example

1. Download and install a C++ compiler;
2. After properly connecting the device, open Matlab. Copy the current path of the "Windows\HTRA_H1P_API_Example\HTRA_Matlab_Examples" folder from the provided USB drive into Matlab's file path box and press Enter;
3. Click the SWPMode_Standard.m file on the left, and confirm whether the compiler address in the first line of the example is consistent with the compiler path installed on the local system;

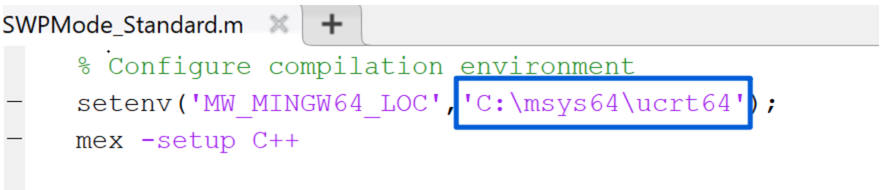

Figure 18 Confirm the compilatopm address

4. Click "Run". The appearance of the Figure 1 window indicates successful execution of the example. For functional descriptions of each example, please refer to the Introduction to Accompanying Examples section in the accompanying materials.

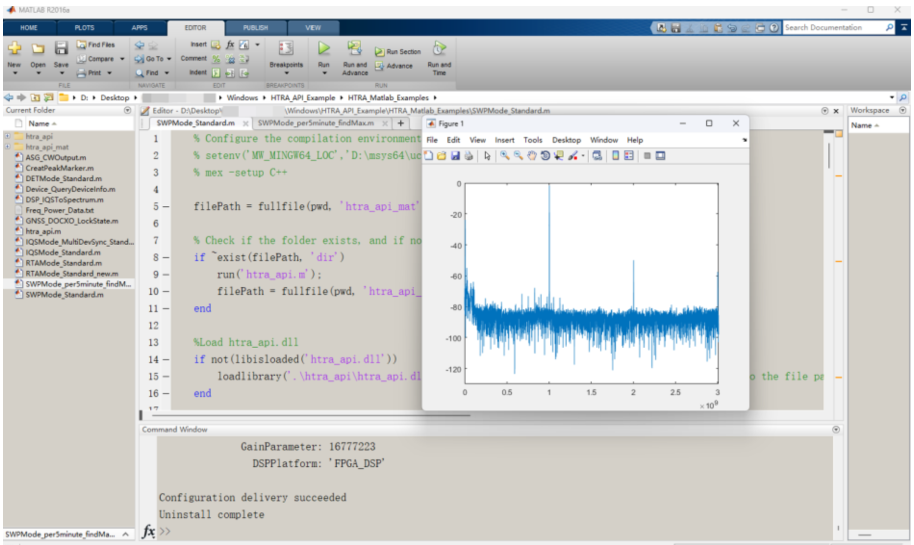

Figure 19 Run the SWPMode_Standard.m File

Note: Since calling the dynamic link library requires a C++ compiler, it is recommended to use MSYS2 to install the g++ compiler. Refer to https://www.msys2.org/ for download and installation instructions.

5.2 Introduction to Accompanying Examples

5.2.1 Get device information

Device_QueryDeviceInfo.m: Retrieve device information, including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

5.2.2 Obtain Standard Spectrum Data

SWPMode_Standard.m: Obtain complete spectrum data within the specified frequency band.

5.2.3 Create multiple markers to display frequency and power of the markers

CreatPeakMarker.m: Obtain spectrum data within the specified frequency band, create markers, and perform peak searching.

5.2.4 Collect the peak spectrum every five minutes

SWPMode_per5minute_findMax.m: Obtain spectrum data within the specified frequency band and search for the peak globally every five minutes.

5.2.5 Obtain continuous stream or fixed number of IQ data

IQSMode_Standard.m: Acquire IQ data under different trigger modes in IQS mode.

5.2.6 The acquired IQ data is converted into spectrum data

DSP_IQSToSpectrum.m: After acquiring IQ data, the obtained IQ data is converted into spectrum data.

5.2.7 Acquire continuous stream or fixed number of power detection data

DETMode_Standard.m: Acquire power detection data under different triggering modes in DET mode.

5.2.8 Acquire continuous stream or fixed duration real-time spectrum data

RTAMode_FixedPoints.m: Acquire real-time spectrum data under different triggering modes in RTA mode.

5.2.9 Acquiring and Plotting Real-Time Spectrum Data in Continuous Streaming or Fixed-Duration Mode

RTAMode_Standard_3D.m: Configures different trigger modes in RTA mode, acquires real-time spectrum data, and plots both the real-time spectrum and probability density displays.

5.2.10 Lock GNSS antenna and DOCXO oscillator

GNSS_DOCXO_LockState.m: Call the API interface to lock the GNSS antenna and DOCXO oscillator, applicable only to devices with IO expansion board options.

5.2.11 Multi-machine synchronization

IQSMode_MultiDevSync_Standard.m: When using the same reference clock source input and the same trigger source input, two devices simultaneously acquire IQ data, allowing for the observation of the synchronization of their collected data.

6. C#

6.1 Configure Development Environment

6.1.1 Development Environment Confirmation

Open the Visual Studio Installer, select the .NET Desktop Development and Universal Windows Platform Development components, then click Modify to ensure that Visual Studio 2019 includes the C# development environment.

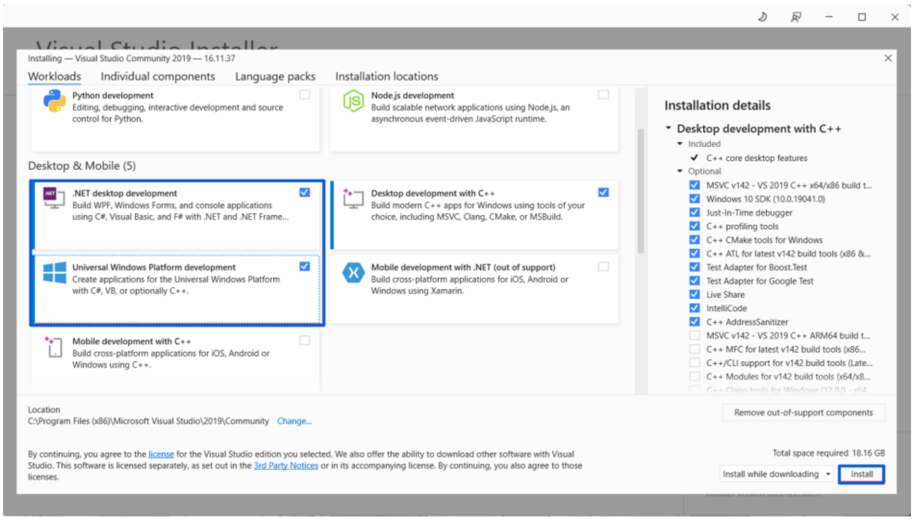

Figure 20 Configuring the C# Development Environment in Visual Studio

6.1.2 Project Setup

1. Open Visual Studio 2019 and click "Create a new project";
2. Select "C# Console Application" and click "Next";
3. Enter the project name and location, uncheck "Place solution and project in the same directory." Select ".NET Framework 4.5" as the framework, and then click "Create";
4. After the project is created, open the project, right-click the solution, and select "Add New Project";
5. In the "Add New Project" dialog, select "Library" as the project type, then choose "Class Library (Universal Windows)" under C#, and click "Next";

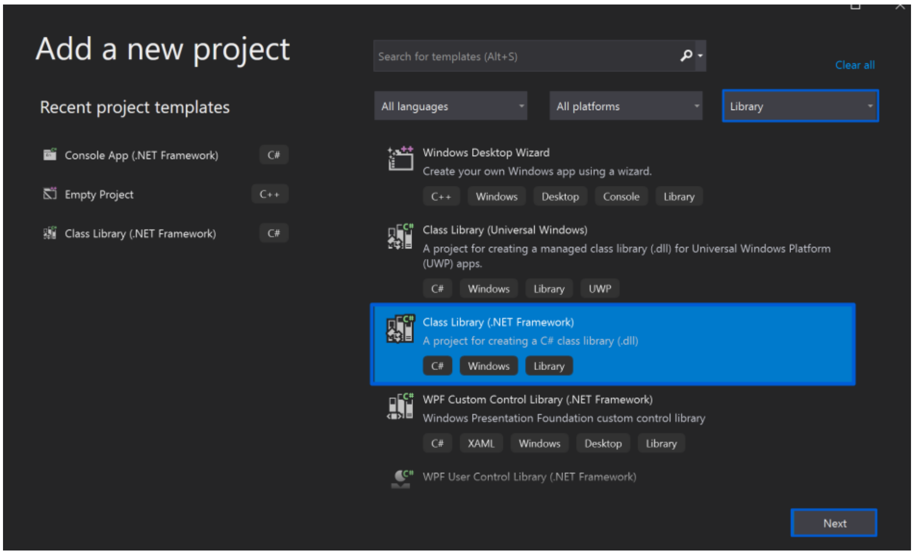

Figure 21 Add New Project

6. Modify the project name as needed (e.g., HtraApi), keep the default location, select ".NET Framework 4.5" as the framework, and click "Create";
7. Right-click ConsoleApp1 and select "Properties". In the project properties window, click the "Build" tab and change the "Platform target" to "x86". Then click the "Debug" tab, enter "htra_api" in the "Working directory" field, click "OK" in the popup, and finally save and close the properties window;

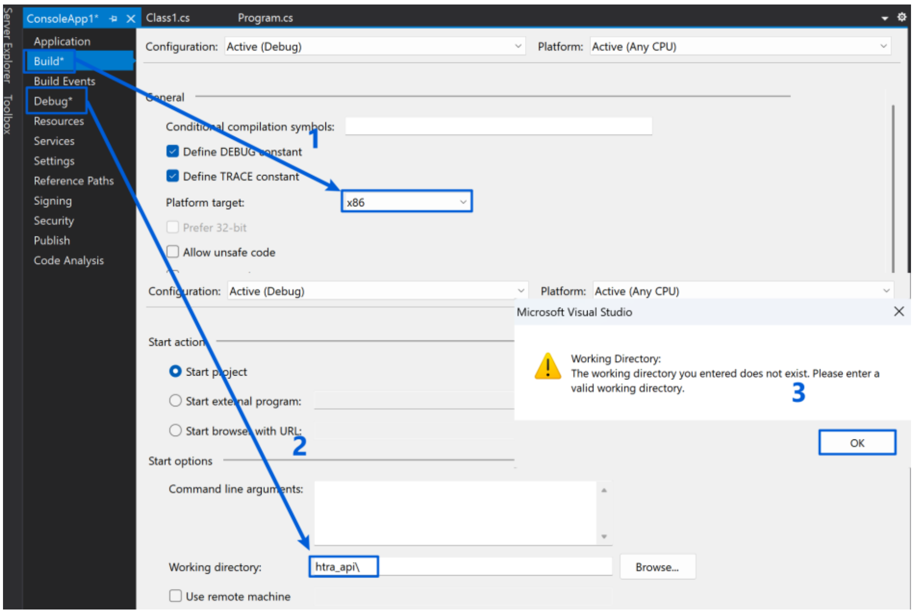

Figure 22 Configure Project Properties

8. Right-click the library HtraApi and select "Properties". In the library properties window, click the "Build" tab, change the "Platform target" to "x86," and save;
9. Copy the contents of the HtraApi.cs file from the "\Windows\HTRA_H1P_API_Example\HTRA_CSharp_Examples " folder on the provided USB drive into the Class1.cs file in the project library, replace the code, and save;

10. Select the ConsoleApp1 project, right-click "References," choose "Add Reference," check the HtraApi library in the project, click "OK," and confirm that the library reference has been added;

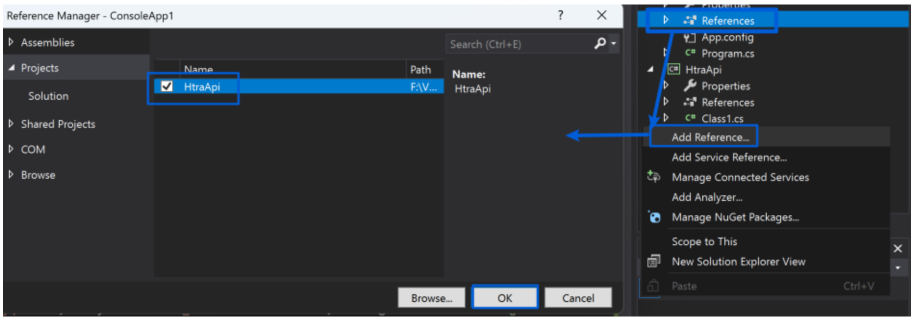

Figure 23 Add Library Reference

11. Copy the htra_api folder from the "\Windows\HTRA_H1P_API\x86" directory on the provided USB drive to the "bin\Debug" folder of the project, and ensure that the "Htra_api\CalFile" folder contains the calibration files for the instruments being used;

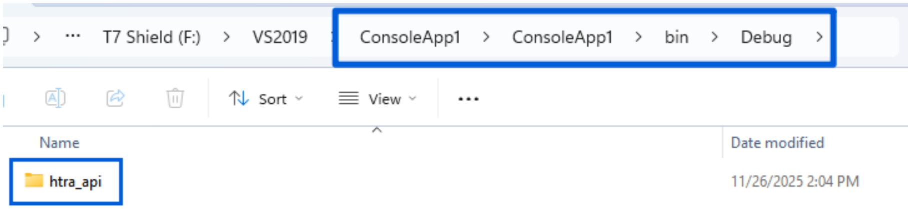

Figure 24 Copy the htra_api file

12. You can refer to the C# examples on the provided USB drive and write code as needed;

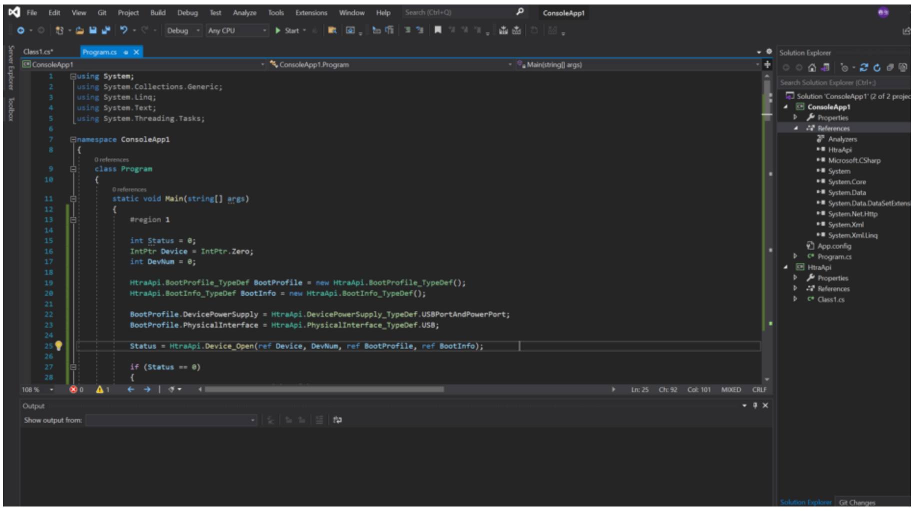

Figure 25 Write C# example

6.2 C# Example Usage Process

1. Open the solution HTRA_CSharp_Examples.sln in Visual Studio from the "Windows\HTRA_H1P_API_Example\HTRA_CSharp_Examples" folder on the provided USB drive;
2. Click the HTRA_CSharp_Examples project on the right, then open the Program.cs file within it;

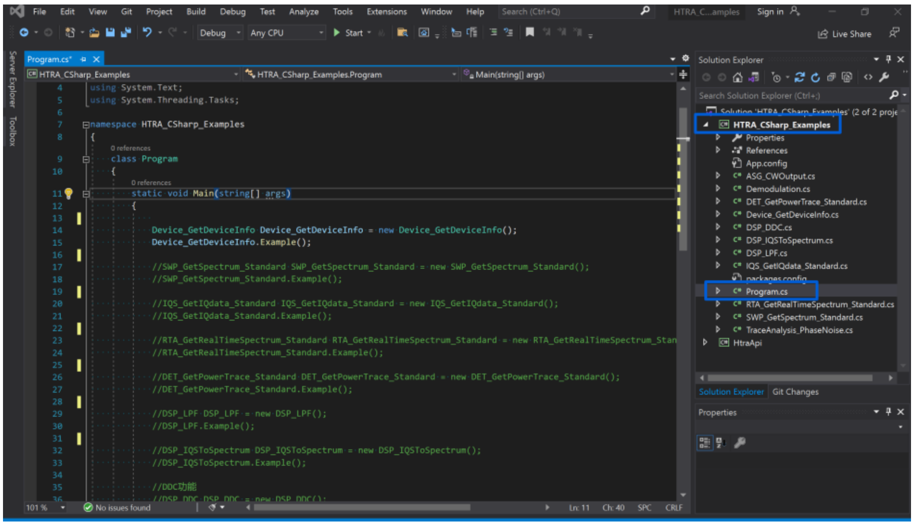

Figure 26 Open the Programe.cs file

3. Each routine in the provided C# examples is encapsulated in a separate class. To use a routine, simply uncomment it (multiple examples cannot be used simultaneously).

   For example, to use the Device_GetDeviceInfo routine, uncomment it and run the program. A successful run is illustrated as follows;

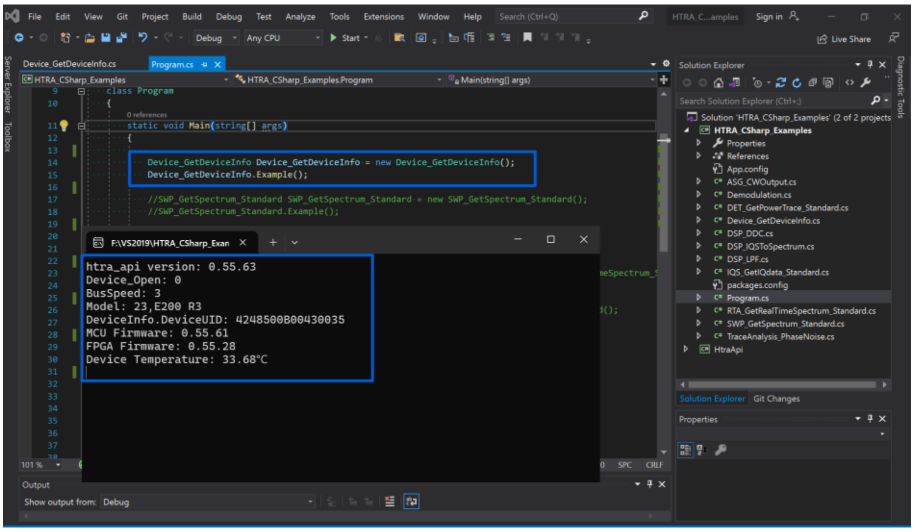

Figure 27 Run Device_GetDeviceInfo.cs

6.3 C# Example Descriptions

6.3.1 Get device information

Device_GetDeviceInfo.cs: Retrieves device information including API version, USB version, device model, device UID, MCU version, FPGA version, and device temperature.

6.3.2 Obtain Standard Spectrum Data

SWP_GetSpectrum_Standard.cs: Obtains complete spectrum data within a specified frequency band.

6.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQdata_Standard.cs: Acquires IQ data under different trigger modes in IQS mode.

6.3.4 Obtain Power Detection Data for a Fixed Number of Points or Duration

DET_GetPowerTrace_Standard.cs: Obtains power detection data under different trigger modes in DET mode.

6.3.5 Obtain real-time spectrum data for a fixed number of points or duration

RTA_GetRealTimeSpectrum_Standard.cs: Retrieves real-time spectrum data under different trigger modes in RTA mode.

6.3.6 Output single-tone signal

ASG_CWOutput.cs: Devices with signal source functionality options output single-tone signals, frequency sweep signals, or power sweep signals through ASG functionality.

6.3.7 IQ to Spectrum Data

DSP_IQSToSpectrum.cs: Converts the IQ data obtained in IQS mode into spectrum data.

6.3.8 Low-pass filtering

DSP_LPF.cs: Applies low-pass filtering to the acquired IQ data and converts it to spectrum.

6.3.9 Digital Downconversion

DSP_DDC.cs: Performs digital down-conversion on the acquired IQ data and converts it to spectrum.

6.3.10 Phase noise test

TraceAnalysis_PhaseNoise.cs: Demonstration of Phase Noise Test Function.

7. Labview

7.1 Configure Development Environment

7.1.1 Using the API in the LabVIEW Environment

1. Open LabVIEW, click "Create Project", select "Project", and click "Finish". An untitled empty project will appear. Press "Ctrl+S" to save the project, choose the save path, enter a project name (e.g., HTRA_Labview_Examples), and click "OK";
2. Copy the "htra_api" folder from the "\Windows\HTRA_H1P_API_Example\HTRA_Labview_Examples\
htra_api " directory on the USB drive to the same directory level as the project. Additionally, you may create an "Examples" folder to store example VIs. If needed, you can also create another folder such as "Subvi" to store sub-VIs;

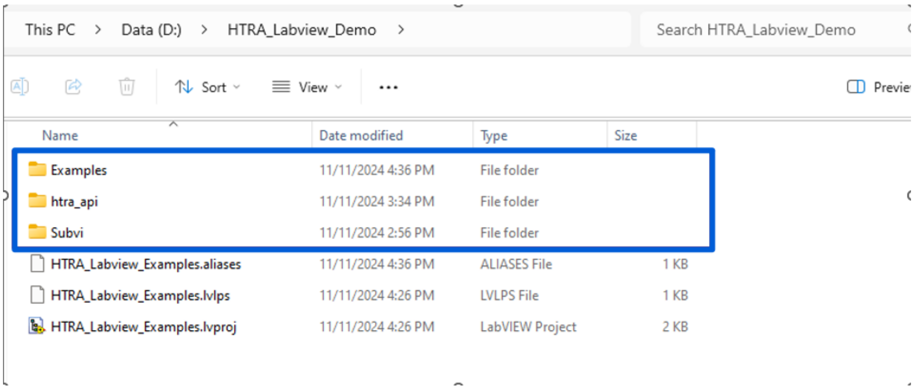

Figure 28 Copy htra_api

3. In the LabVIEW project, add the newly created Examples folder and the htra_api folder to the project, then save it;

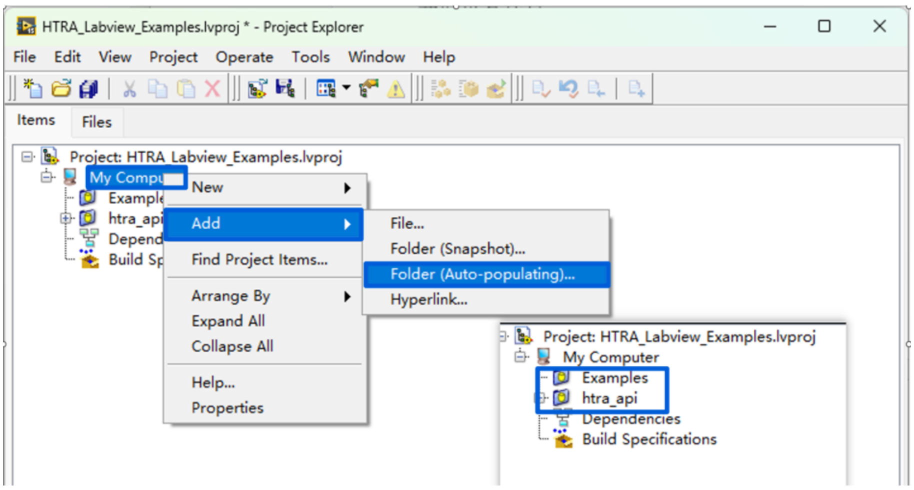

Figure 29 Add to project

4. Create a new VI in the Examples folder. In the block diagram, you can call the exported

LabVIEW API functions. The calling process is the same as in the C environment;

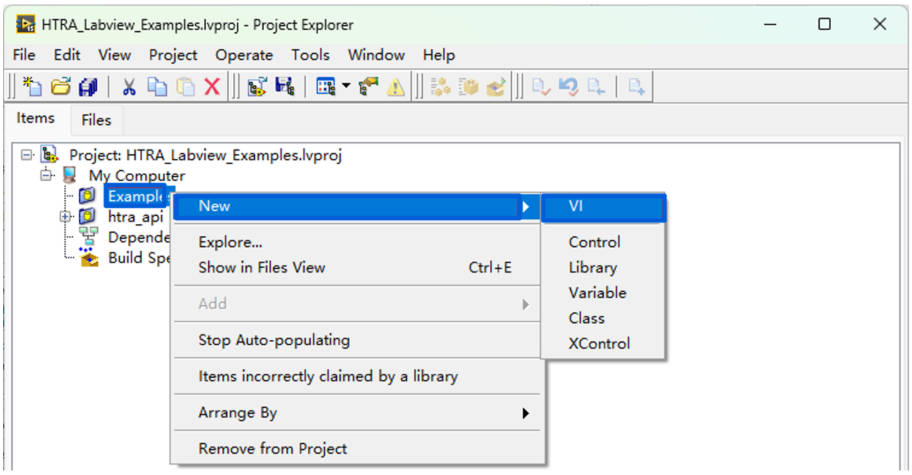

Figure 30 Create VI

5. Finally, save the VI to the Examples folder and rename it as needed.

7.2 Labview Example Usage Process

The usage process of Labview examples included in the USB drive is as follows:

1. Use LabVIEW to open the project "HTRA_Labview_Examples.lvproj" located in the USB drive under Windows\HTRA_H1P_API_Example\HTRA_Labview_Examples;

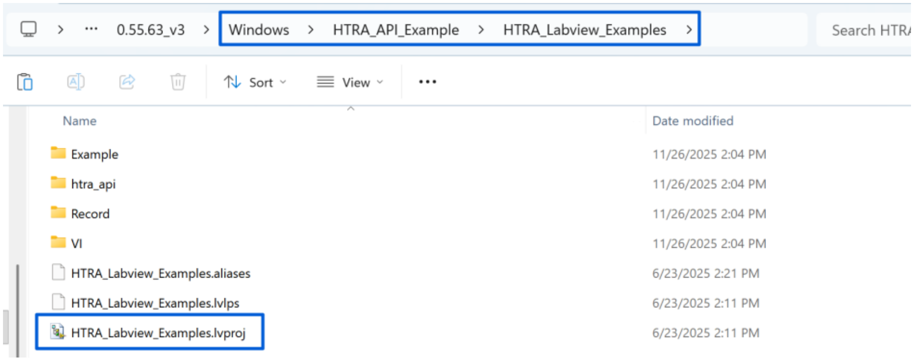

Figure 31 Open Labview Project

2. Double-click any example VI in the Example folder. Here, SWP_GetSpectrum_Standard.vi is used as an example;

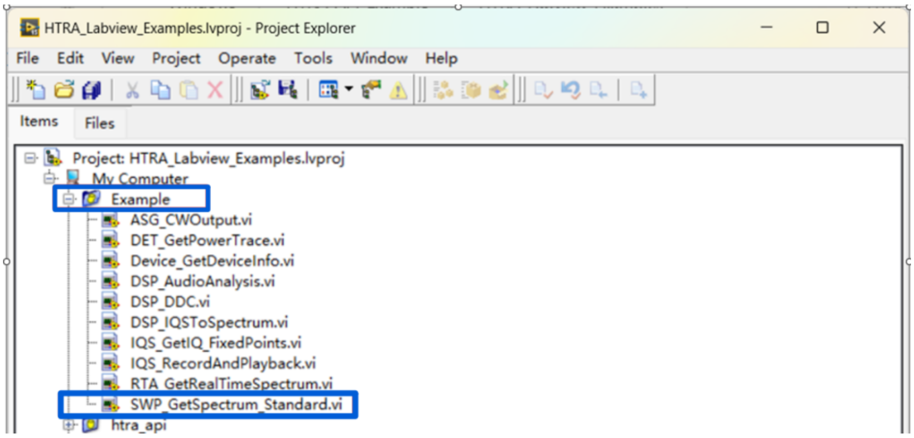

Figure 32 Open example

3. After opening the VI, click the Run button in the upper-left corner to execute the program. The running interface is shown below;

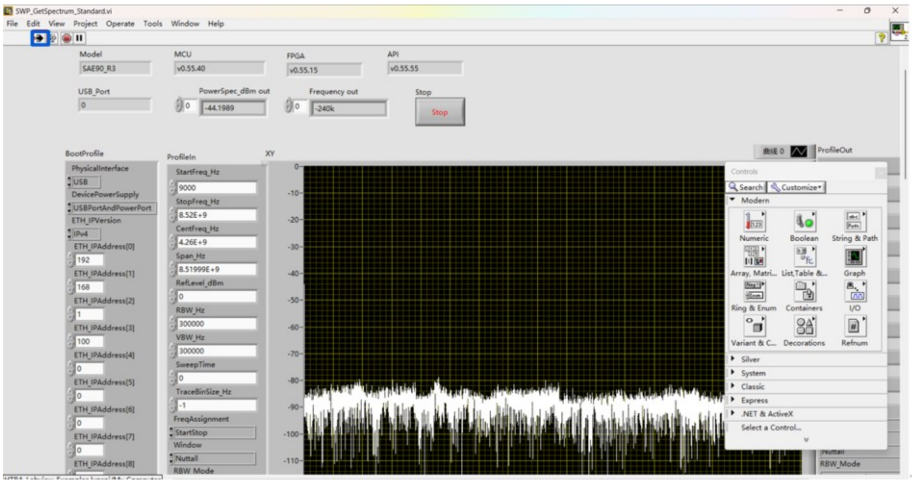

Figure 33 Successfully Run the SWP_GetSpectrum_Standard Example

7.3 Labview Example Description

7.3.1 Get device information

Device_GetDeviceInfo.vi: An example for obtaining various device information, including: API version, device model, device UID, MCU version, FPGA version, and device temperature.

7.3.2 Standard spectrum acquisition

SWP_GetSpectrum_Standard.vi: Obtains standard spectrum data within a specified frequency band and displays the image.

7.3.3 Obtain IQ Data for a Fixed Number of Points or Duration

IQS_GetIQ_Standard.vi: Obtains IQ data with a fixed number of points. When the device receives a Bus trigger signal, it returns IQ data with a fixed number of points.

7.3.4 Streaming and reading IQ data

IQS_RecordAndPlayback.vi: Obtains IQ data, records the IQ data as a txt file, and plays back the recorded IQ data.

7.3.5 IQ to Spectrum Data

DSP_IQSToSpectrum.vi: Obtains IQ data and converts the acquired IQ data into spectrum data.

7.3.6 Digital Downconversion

DSP_DDC.vi: Performs digital downconversion on the IQ data stream and resamples to generate a sub-IQ stream for further spectrum analysis.

7.3.7 Audio Analysis

DSP_AudioAnalysis.vi: Analyzes audio voltage (V), audio frequency (Hz), signal-to-noise ratio (dB), and total harmonic distortion (%).

7.3.8 Obtain Power Detection Data for a Fixed Number of Points or Duration

DETMode_Standard.vi: Obtain a fixed number of DET data points. When the device receives a Bus trigger signal, it returns a fixed number of DET data points.

7.3.9 Obtain real-time spectrum data for a fixed number of points or duration

RTAMode_Standard.vi: Obtain a fixed number of RTA data points. When the device receives a Bus trigger signal, it returns a fixed number of RTA data points.

7.3.10 ASG Signal Source Output Signal

ASG_SignalOutput.vi: Control the internal signal generator of the device to output single-tone signals, sweep signals, and power scan signals.

8. Linux

8.1 Environment Version Compatibility Self-Check

When using the device in a Linux system, you first need to confirm whether the current Linux environment's system architecture, gcc version, and GLIBC version are supported according to the following process:

1. Open the terminal and enter *uname -a* to check the Linux system architecture, for example, here the Linux system architecture is x86_64;
2. In the terminal, enter *gcc -v* to check the system gcc version, for example, here the gcc version is 7.5.0;
3. In the terminal, enter *ldd --version* to check the system GLIBC version, for example, here the GLIBC version is 2.27;

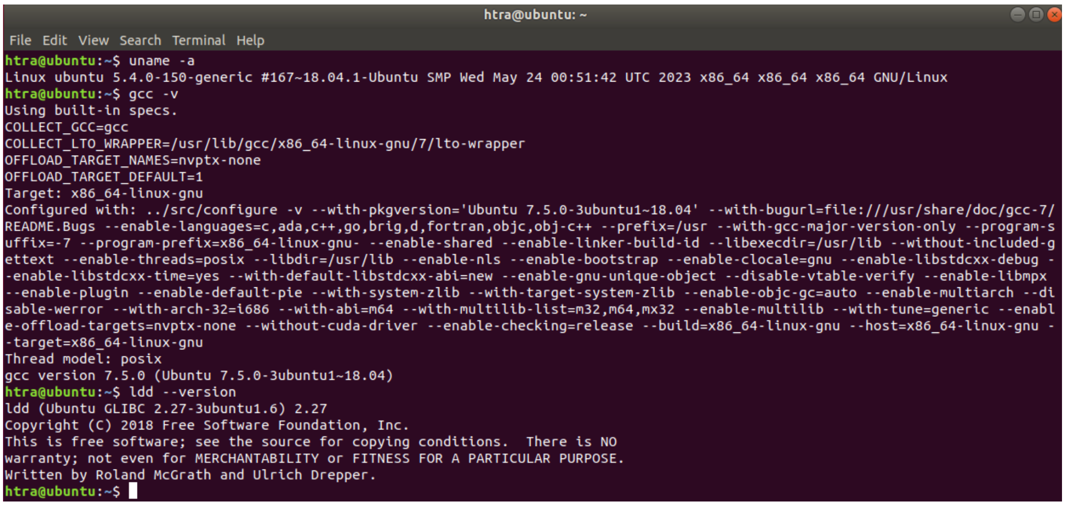

Figure 34 Check the Linux System Architecture and Version

4. Compare the terminal information with the table to confirm whether the current environment is supported. If it is not supported, please contact technical support.

Table 1 Terminal Information Comparison Table

x86 processor	Support Intel and AMD processors
ARM processor	aarch64 (armv8), armv7 processors, such as: Raspberry Pi 4b, RK3399, RK3568, RK3588, T507, NVIDIA Jetson TX2
Compilation environment	Gcc7.5, glib2.27 and above
Distribution	Customized system for Raspberry Pi 4b, Ubuntu 18.04, etc.

8.2 Accompanying documentation

Currently, the Linux section of the accompanying USB drive contains the following materials:

8.2.1 HTRA_C++_Examples

The HTRA_C++_Examples folder contains:

1. Examples folder: C++ example programs (see the C++ Example Usage section for instructions);
2. Makefile: Build script used to compile the example programs into executables;
3. bin folder: Stores device calibration files and the executables generated from building the example programs.

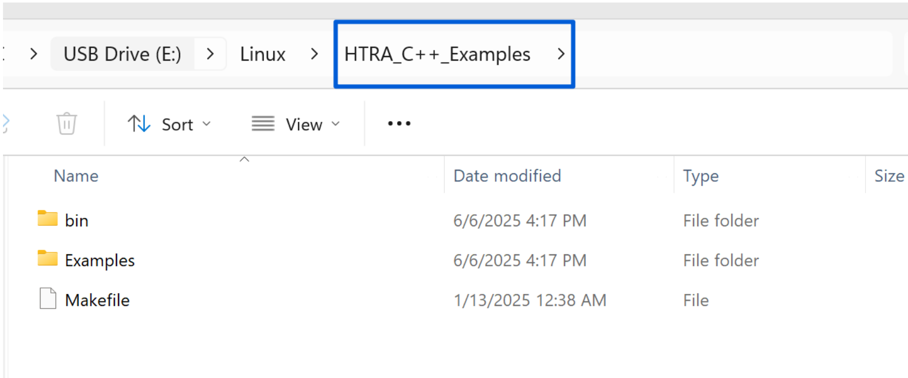

Figure 35 Contents of the HTRA_C++_Examples Folder on Linux

8.2.2 HTRA_Qt_Examples

The HTRA_Qt_Examples folder contains:

1. htrademo folder: Qt examples and .pro files (see the Qt Example Usage section for instructions);
2. bin folder: Stores device calibration files and executables generated from compiling the example programs;
3. htraapi folder: Stores the dynamic link libraries.

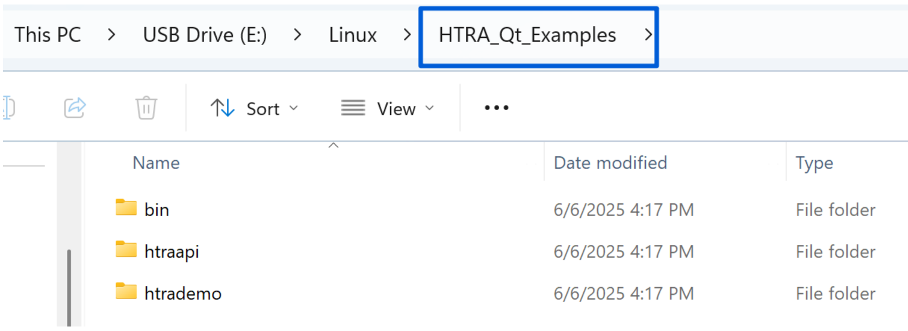

Figure 36 Contents of the HTRA_Qt_Examples Folder on Linux

8.2.3 HTRA_Python_Examples

The HTRA_Python_Examples folder specifically contains:

1. Python example programs (see the Python Example Usage section for instructions);
2. CalFile folder: Stores device calibration files;
3. Htraapi folder: Stores the dynamic link libraries.

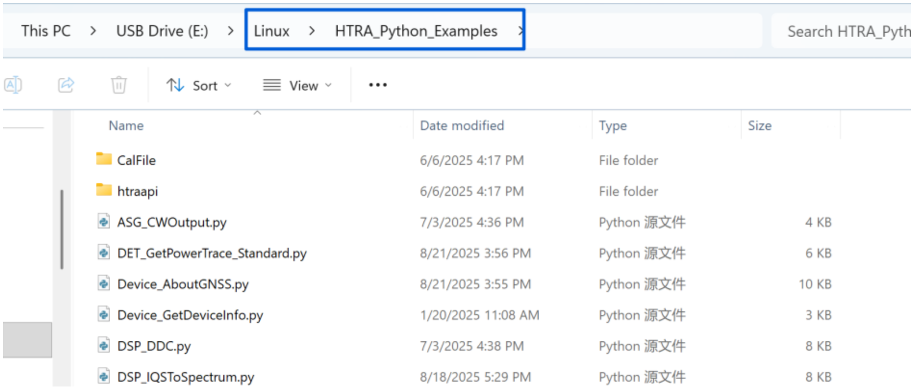

Figure 37 Contents of the HTRA_Python_Examples folder in Linux

8.2.4 Install_HTRA_SDK

1. The Install_HTRA_SDK folder contains:
   - install_htraapi_lib.sh: Driver configuration script
   - htraapi folder: Stores drivers, header files, and libraries
2. The Install_HTRA_SDK/htraapi folder contains:
   - configs folder: Driver configuration files
   - inc folder: Header files
   - lib folder: Dynamic link libraries
3. The Install_HTRA_SDK/htraapi/lib folder contains:
   - arrch64 folder: ynamic link libraries for the arrch64 architecture
   - x86_64 folder: Dynamic link libraries for the x86_64 architecture
   - armv7 folder: Dynamic link libraries for the armv7 architecture

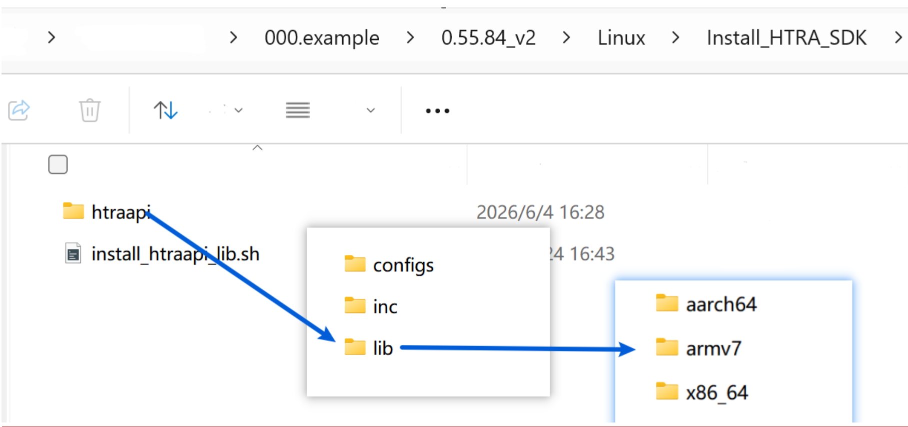

Figure 38 Contents of the Install_HTRA_SDK Folder on Linux

8.3 Driver File Configuration

To use the device in Linux, the driver files must be configured first. The specific procedure is as follows:

1. Configure the driver files: First, copy the Install_HTRA_SDK folder to the Linux host. Then, open a terminal in the Install_HTRA_SDK folder and enter *sudo sh install_htraapi_lib.sh* to configure the driver files.

Note: If the development board does not have the sudo command, simply enter *sh install_htraapi_lib.sh*.

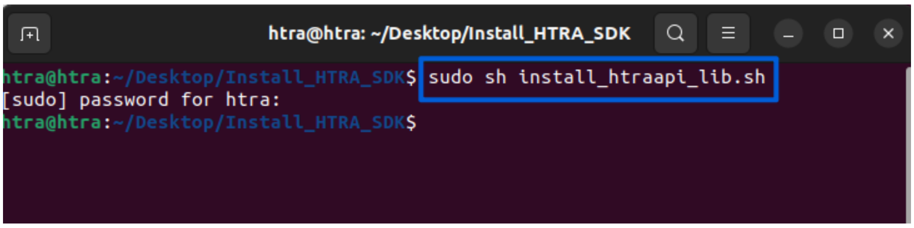

Figure 39 Configure driver files on Linux

2. Configuration check: Ensure the device is correctly connected to the host computer (if the host computer is a virtual machine, ensure the device is connected to the virtual machine and that USB compatibility is 3.1) and provide normal power to the device. At this point, as shown in the figure, enter *lsusb* in the terminal to view the list of USB devices on the machine, where "ID: 6430 (or ID: 3675 or ID: 04b5)" indicates successful device connection.

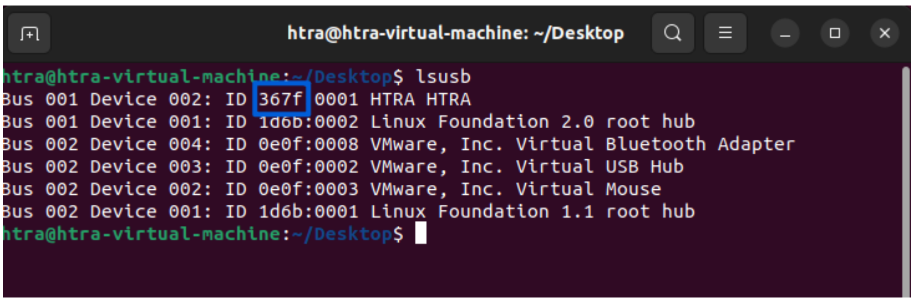

Figure 40 Check device connection status on Linux

8.4 C++ Example Usage and Project Creation

8.4.1 C++ Example Usage

Prerequisite: Ensure the device is properly connected and the driver files have been correctly configured as described in the Driver File Configuration section.

You can follow the process below and use the C++ examples provided on the USB drive (the specific functions of the examples can be found in the corresponding C/C++ sections).

1. Select the program to compile: First, copy the "Linux\HTRA_C++_Examples" folder from the provided USB drive to the host computer. Then, open main.cpp in the Examples folder and uncomment the examples you want to test or use;

Figure 41 Uncomment the SWP_GetSpectrum_Standard Example

2. Compile the selected example: First, check the system architecture according to the Environment Version Compatibility Self-Check section. Open a terminal in the HTRA_C++_Examples folder and enter the corresponding command based on the host system architecture (the following example shows how to compile the SWP_GetSpectrum_Standard routine).
   - For x86_64 system:

     make Example=SWP_GetSpectrum_Standard
   - For aarch64 system:

     make TARG=aarch64 Example=SWP_GetSpectrum_Standard
   - For armv7 system:

     make TARG=armv7 Example=SWP_GetSpectrum_Standard

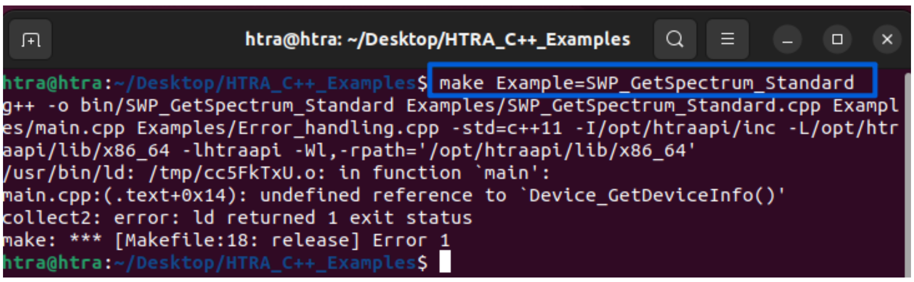

Figure 42 Compile SWP_GetSpectrum_Standard on x86_64 System

3. Verify the calibration files: Go to the "HTRA_C++_Examples\bin\CalFile" folder and ensure

it contains the calibration files for the devices being used *(Skip this step for SAN-45 R6, SAN-60 R6, and SAN-90 R0 devices.)*;

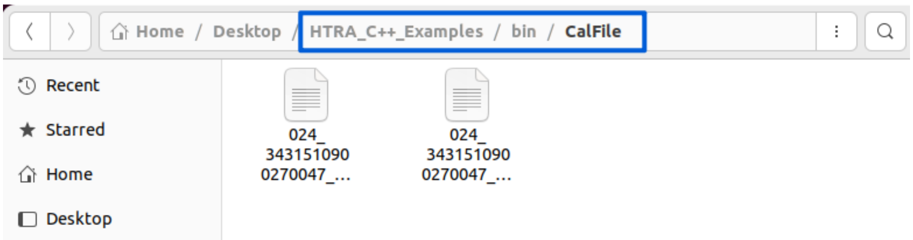

Figure 43 Verify the calibration files

4. Run the program: After successfully compiling the program and verifying the calibration files, open a terminal and enter:

   ./bin/SWP_GetSpectrum_Standard to run the selected example.

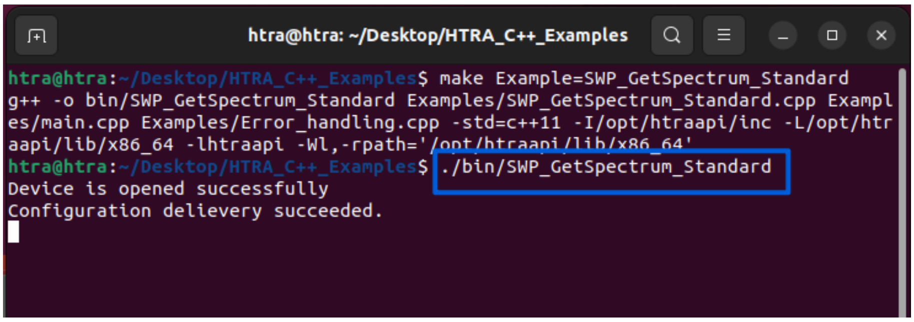

Figure 44 Run the Compiled SWP_GetSpectrum_Standard Example

8.4.2 C++ Project Creation and Compilation

Prerequisite: Correctly configure the driver files as described in the Driver File Configuration section.

1. Write the code:

   The Linux dynamic libraries provided on the USB drive are identical to those on Windows. The code you write only needs to comply with the BNC ICX Series API Programming Guide.

2. Compile and run:

1). Create a new folder to store the entire project (e.g., C++_Test). Inside this folder, create a CalFile folder to store calibration files and an htraapi folder to store header files and dynamic link libraries;

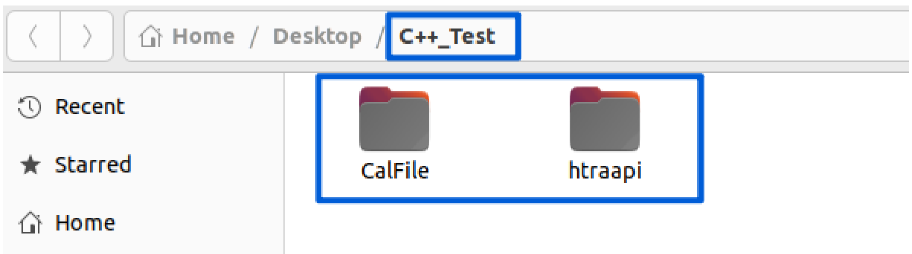

Figure 45 Create Project Folder

2). Inside the htraapi folder, create an inc folder to store header files and a lib folder to store dynamic link libraries;

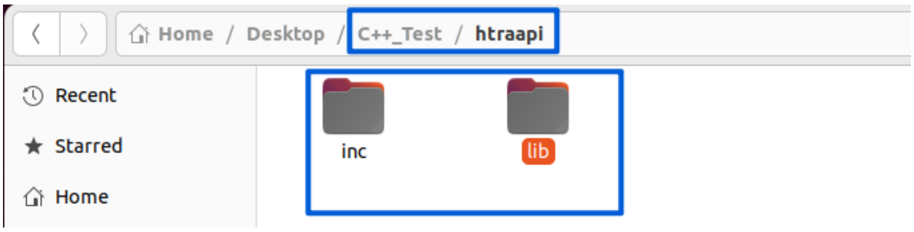

Figure 46 Create inc and lib Folders under htraapi

3). Copy the files from the CalFile folder on the USB drive into the newly created "C++_Test\CalFile" folder *(Skip this step for SAN-45 R6, SAN-60 R6, and SAN-90 R0 devices.)*;

4). Copy the header files from "Linux\Install_HTRA_SDK\htraapi\inc" on the USB drive into the newly created "C++_Test\htraapi\inc" folder;

5). heck the system architecture according to the Environment Version Compatibility Self-Check section. Then, following the description of the lib folder, copy the files from the folder corresponding to your architecture into the "C++_Test\htraapi\lib" folder. (The illustration shows an x86_64 architecture host.);

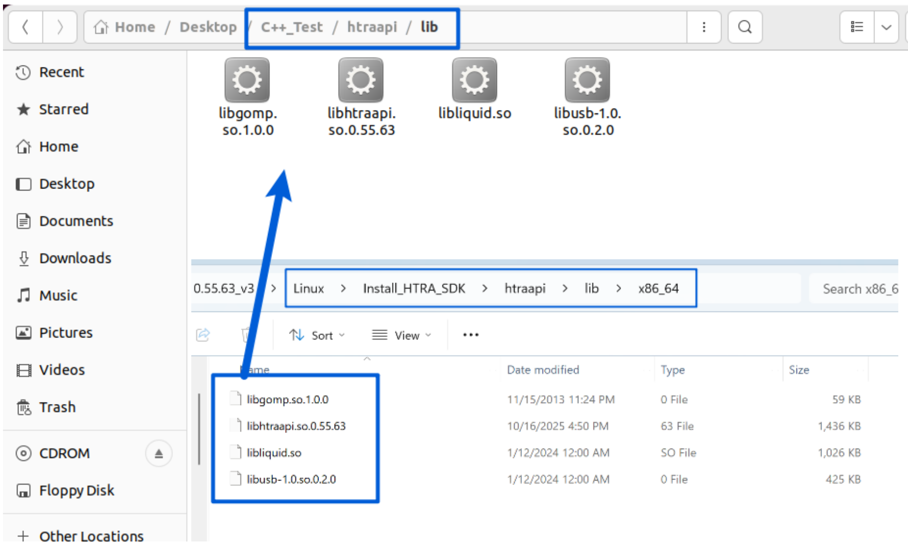

Figure 47 Copy the Dynamic Link Libraries for the Corresponding Architecture into the lib Folder

6). Open a terminal in the lib folder and enter the following command to create symbolic links for the copied dynamic link libraries (the commands are the same for all three architectures): **In the following, the libhtraapi.so library uses version 0.55.63 as an example; for other versions, simply adjust the version number accordingly:**

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

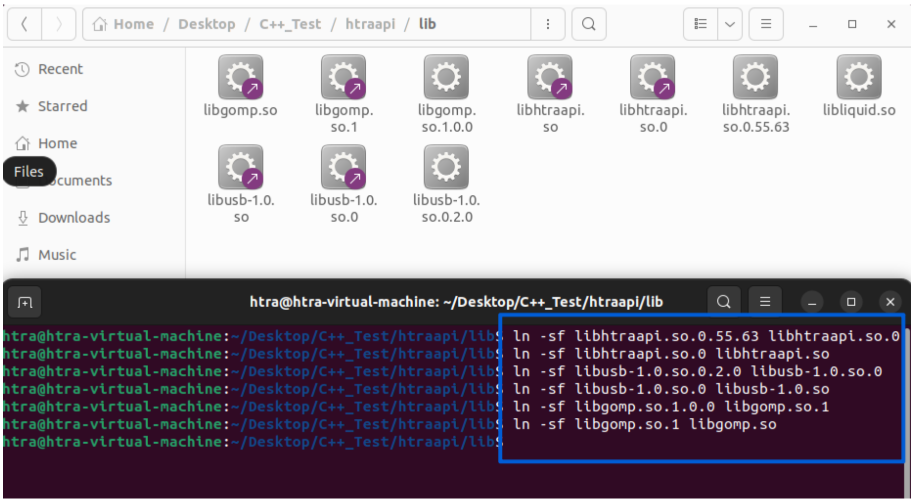

Figure 48 Create Symbolic Links for Dynamic Link Libraries

7). Place the written code files in the top-level folder of C++_Test;

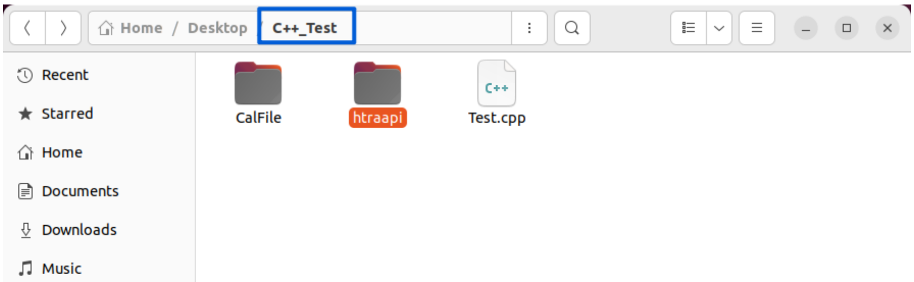

Figure 49 Place the Code Files

8). Compile to generate the executable: Based on the system architecture confirmed in the Environment Version Compatibility Self-Check section, open a terminal in the C++_Test folder and enter the command corresponding to your architecture;

The following example uses Test.cpp. Compile it on different operating systems to generate an executable named Test:

- For x86_64 system:

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

- For arrch64 system:

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

- For armv7 system:

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

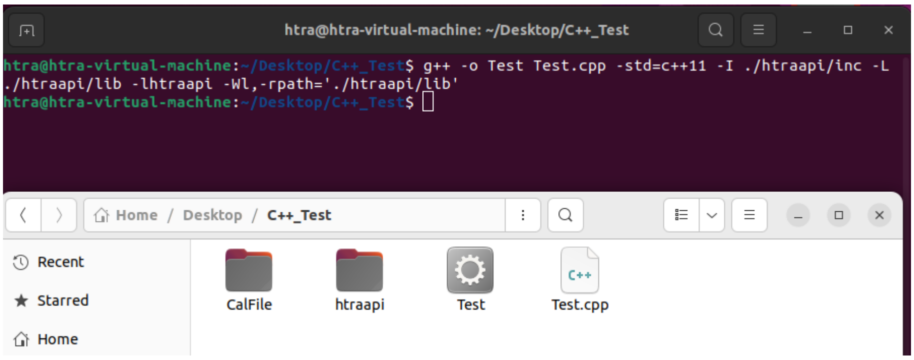

Figure 50 Compile to Generate Executable

9). Run the program: Enter *./Test* to start the executable file.

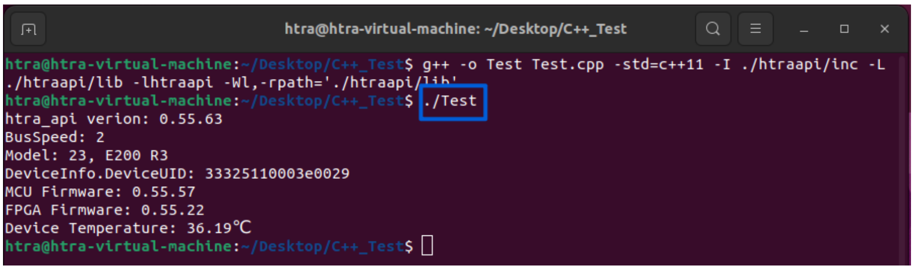

Figure 51 Run the Test Executable

8.4.3 C++ Project Cross Compilation

If the host computer has a cross-compilation toolchain and you want to cross-compile for the device, follow the process below.

The following example shows how to cross-compile an aarch64 executable on an x86_64 host:

1. Generate the executable for the target architecture:

1). Follow steps 1 to 8 in the "C++ Project Creation and Compilation" section to create the project, place the calibration files, and add the aarch64 architecture header and library files. Then, create symbolic links, place the program code, and compile the executable using the target architecture compilation command.

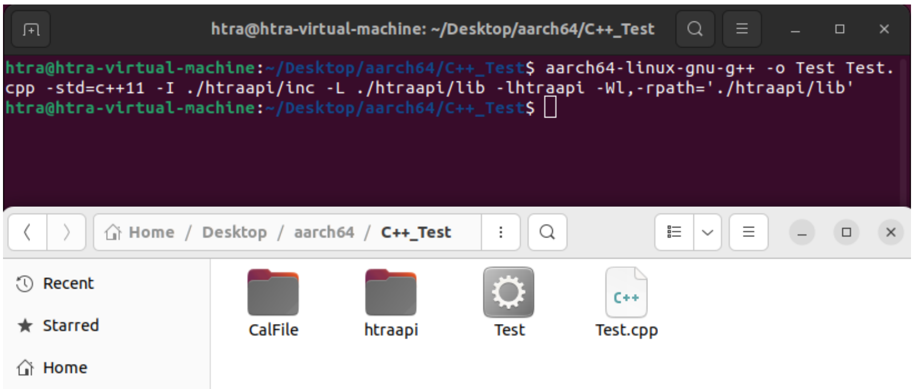

Figure 52 Cross-Compile an aarch64 Executable

2). After generating the executable, enter *file Test* to check the architecture of the executable. You should see that the current executable is for the aarch64 architecture.

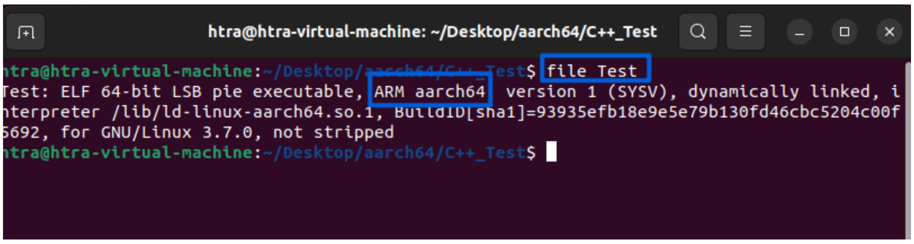

Figure 53 Check Executable Architecture

2. Run the executable on an aarch64 host:

1). Navigate to the project directory and execute *zip -r C++_Test.zip C++_Test* to compress the entire C++_Test folder into a ZIP archive. Then, copy the generated ZIP file to the aarch64 host;

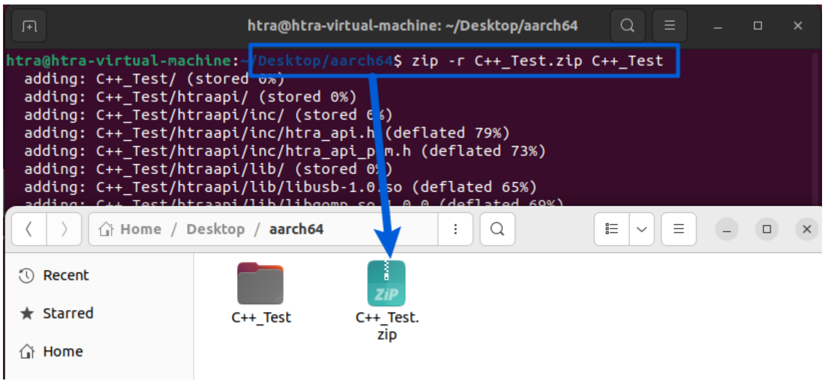

Figure 54 Compress the Created Project

2). On the aarch64 host, enter *unzip C++_Test.zip* to extract the project;

3). Following the Driver File Configuration section, configure the driver files on the aarch64 host;

4). After configuring the driver files, go to the C++_Test folder and run the program by entering ./Test in the terminal.

8.5 Using QT Examples and Project Creation

8.5.1 Qt Example Usage Process

Usage prerequisite: Ensure the device is properly connected and the driver files have been correctly configured according to the Driver File Configuration chapter.

The following example demonstrates how to run the Qt sample on Ubuntu 22.04, using an x86_64 architecture.

1. Copy the "Linux\HTRA_Qt_Examples" folder from the included USB drive to the host PC;
2. Refer to the Environment Version Compatibility Self-Check chapter to confirm the system architecture, then copy the contents from the corresponding architecture folder under "Linux\Install_HTRA_SDK\htraapi\lib" on the USB drive into the host PC's "HTRA_Qt_Examples\htraapi" folder;

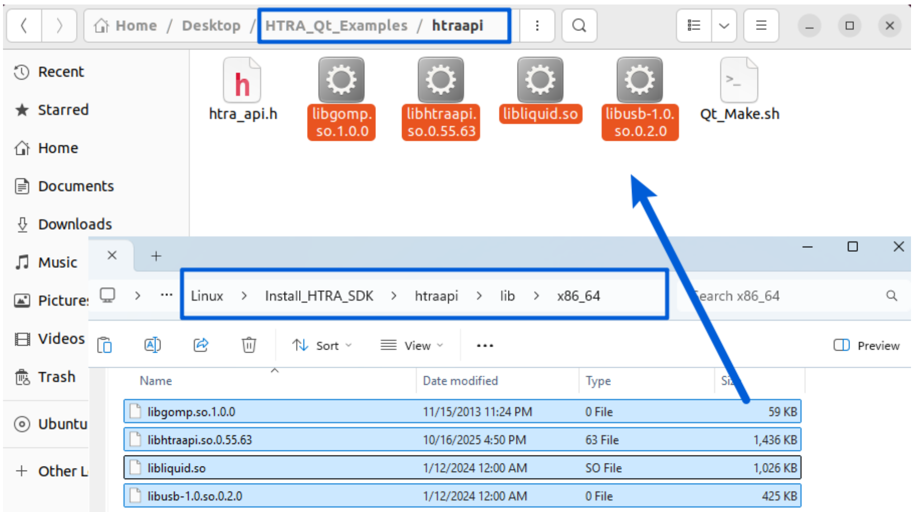

Figure 55 Copy the dynamic libraries for the corresponding architecture

3. Open a terminal in the htraapi folder and enter *sudo sh Qt_Make.sh*. When prompted, enter the password to authorize the creation of symbolic link;

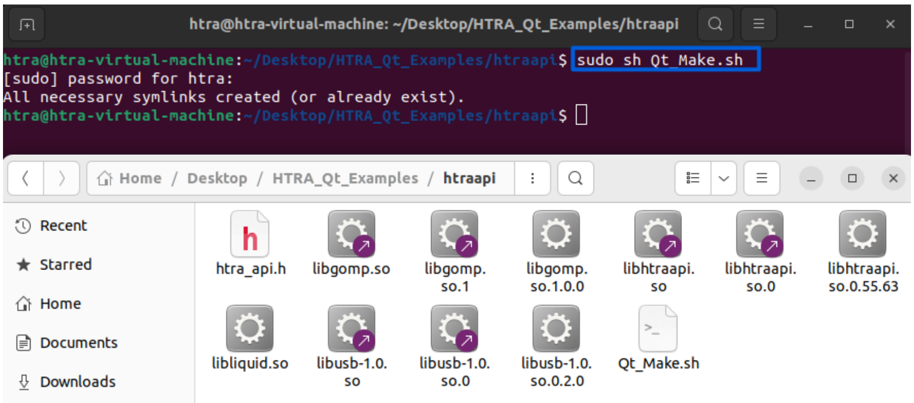

Figure 56 Create symbolic links for the dynamic libraries

4. Enter the "HTRA_Qt_Examples\bin\CalFile" folder and ensure that it contains the calibration files for the device in use;
5. Use Qt Creator to open the htrademo.pro file located in "HTRA_Qt_Examples\htrademo", and configure the project's build environment;

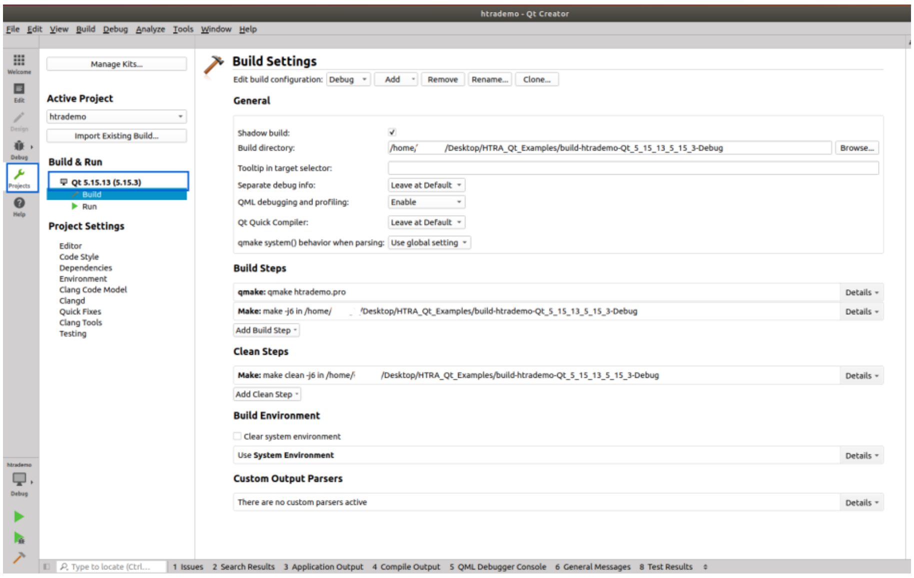

Figure 57 Build environment

6. Click "Edit", open main.cpp under the "Sources" folder of the "htrademo" project, uncomment the relevant functions, save the file, and click Run to execute different examples;

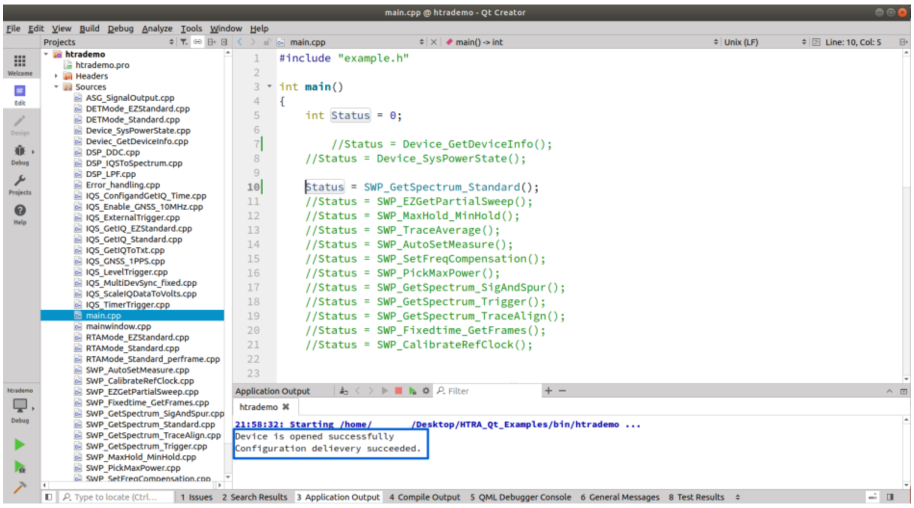

Figure 58 Run the SWP_GetSpectrum_Standard example

8.5.2 Qt Project Creation and Compilation

Provided that the driver files have been correctly configured as described in the Driver File Configuration section, if you want to create and compile a Qt project, follow the process below: The code should follow the API Programming Guide, and the steps for creating a GUI application are as follows:

1. Create a new folder to store the entire project (e.g., QtTest), including bin (for calibration files and generated executables) and htraapi (for header files and dynamic link libraries) subfolders;
2. Copy the instrument-specific "CalFile" folder from the USB drive into the "QtTest\bin" folder;
3. Copy the header files from "Linux\Install_HTRA_SDK\htraapi\inc" on the USB drive into the QtTest\htraapi folder;
4. Copy the dynamic link libraries from the folder corresponding to your system architecture under "Linux\Install_HTRA_SDK\htraapi\lib" on the USB drive into the "QtTest\htraapi" folder (here, an x86_64 host is used as an example);

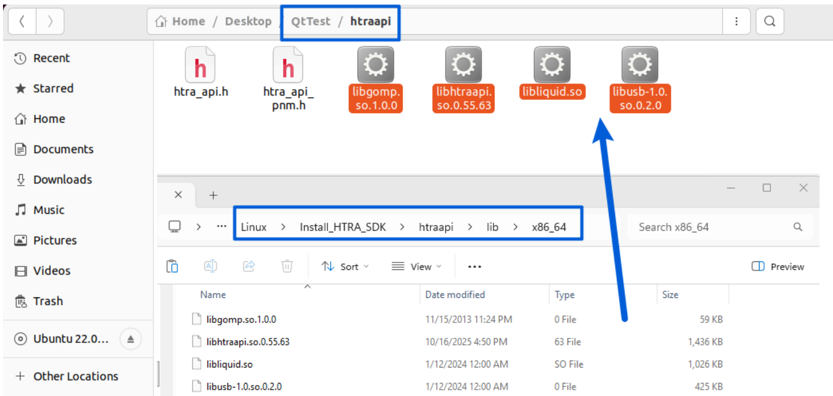

Figure 59 Copy the dynamic link libraries for the corresponding architecture

5. Open a terminal in the htraapi folder and enter the following commands in sequence to create symbolic links for the copied dynamic link libraries (the commands are the same for all architectures):

In the following, the libhtraapi.so library uses version 0.55.63 as an example; for other versions, simply adjust the version number accordingly:

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

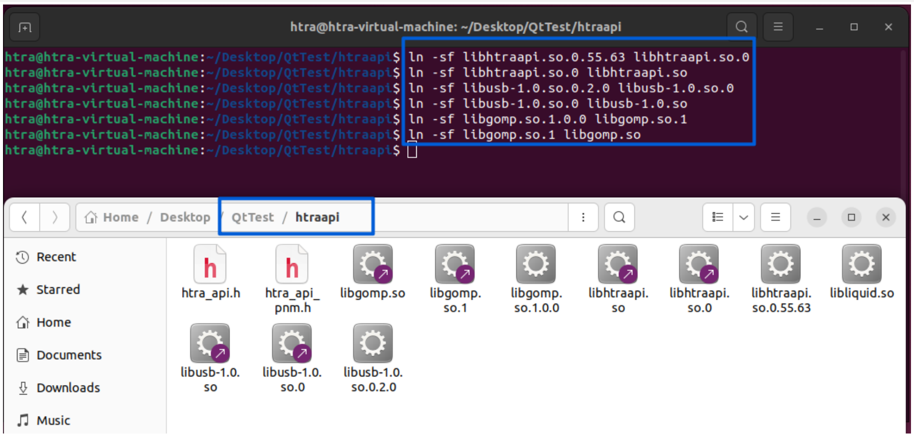

Figure 60 Create symbolic links for the dynamic libraries in htraapi

6. Open Qt Creator and click "File" -> "New File or Project";
7. In the project options, click "Application", select "Qt Widgets Application", and click "Choose…";
8. Enter the project name (e.g., "test"), click "Browse", select the previously created "QtTest" folder, and then click "Next";
9. Select "qmake", click "Next" until the "Kit Selection" screen appears. On this screen, choose a build environment and click "Next";
10. Click "Finish" to create the project;
11. In the Qt Creator main interface, right-click the "Test" project, select "Add Library…" -> "External Library" -> "Next";
12. Click "Browse" for the library file, select libhtraapi.so in "QtTest\htraapi", click "Open", choose the "Linux" platform, and click "Next".

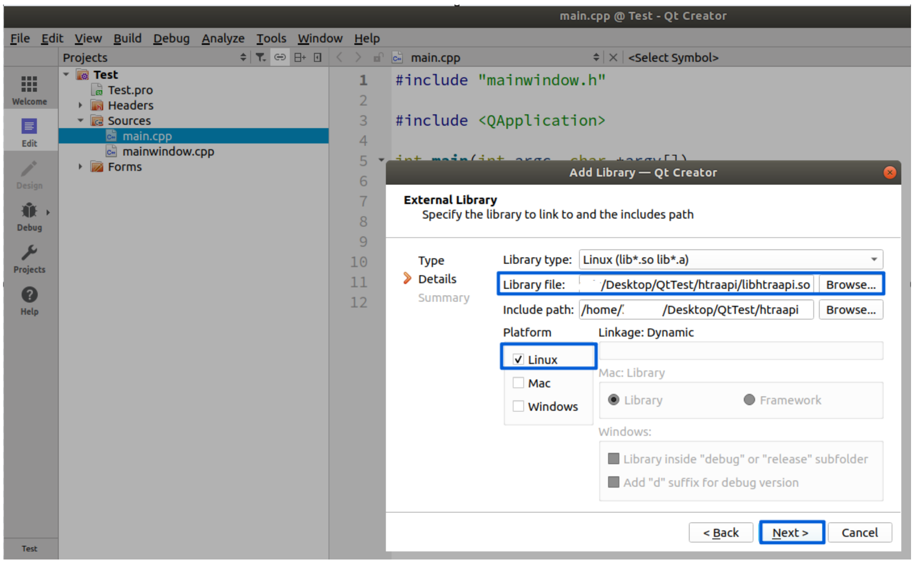

Figure 61 Select the Linux Platform

13. Click "Finish" to add the external library;

14. In Test.pro, after CONFIG += c++11, add *DESTDIR = $$clean_path($$PWD/../bin)* to specify the location for the generated executable;

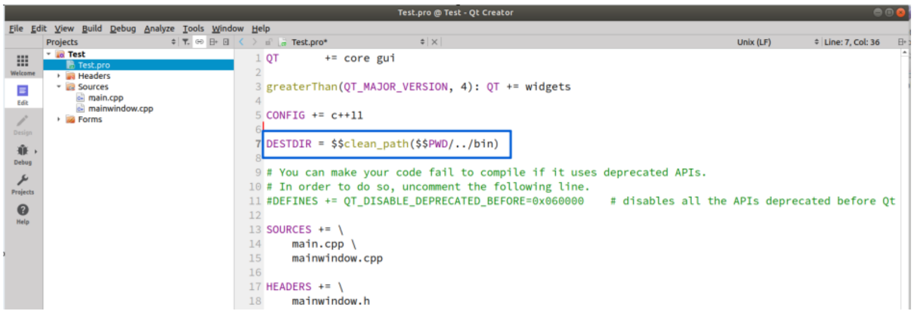

Figure 62 Specify the Executable Output Location

15. After the library file, add *-Wl,-rpath,$$PWD/../htraapi* to specify the runtime library path for the executable;

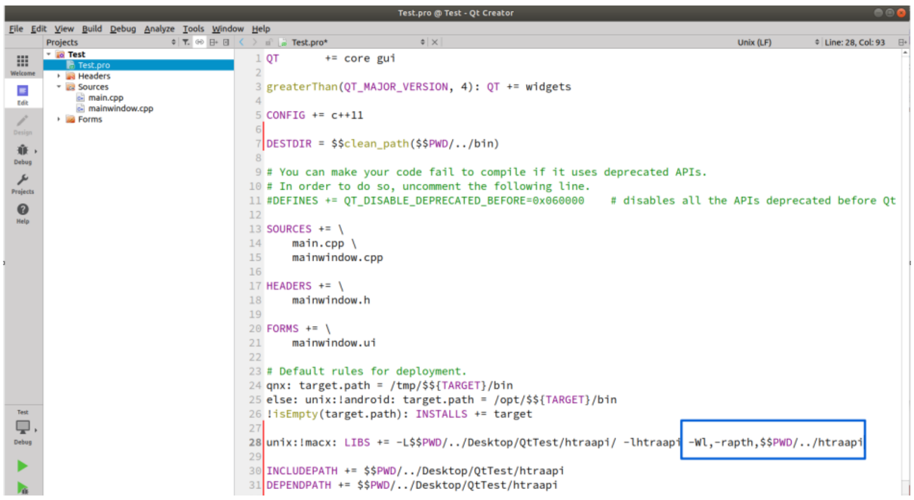

Figure 63 Specify the Runtime Library Path for the Executable

16. Save the Test.pro file, then write the code in mainwindow.cpp and click Run. The normal

operating interface is shown below;

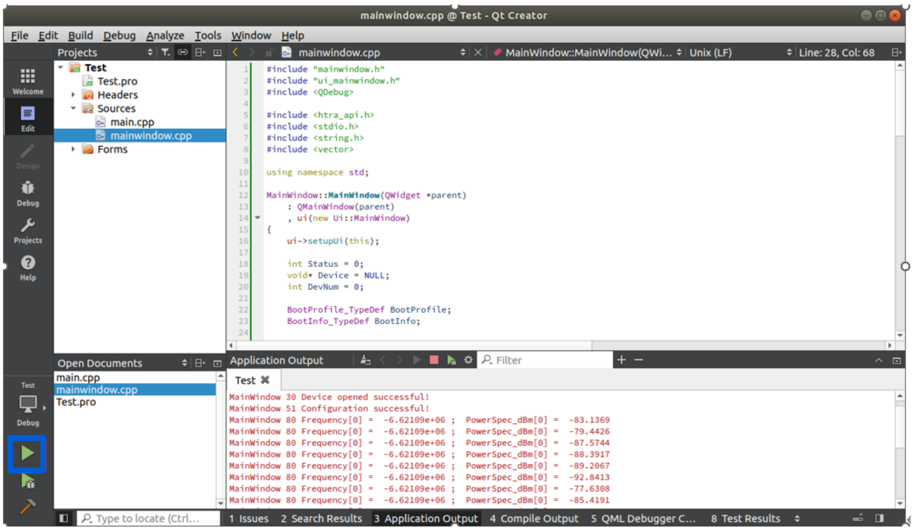

Figure 64 Run the Code

17. Close Qt Creator, navigate to the QtTest\bin folder, open a terminal, and enter *./Test* to run the executable;

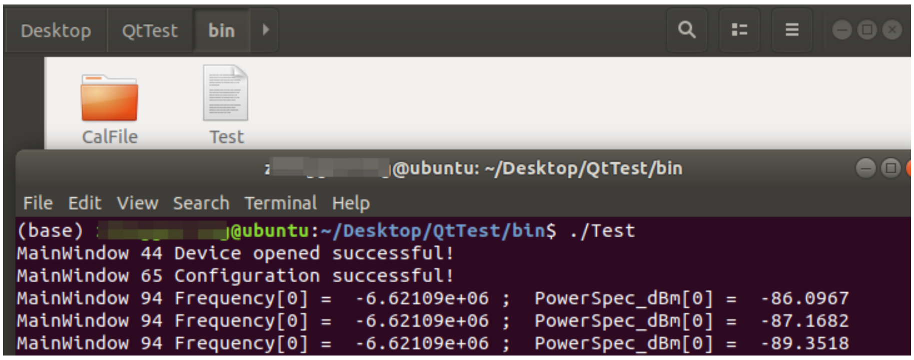

Figure 65 Run the Executable

8.5.3 Qt Project Cross-Compilation

If the host computer has a cross-compilation toolchain and you want to cross-compile for the device, follow the procedure below (here, an example of cross-compiling an aarch64 executable on an x86_64 host is given):

1. First, generate the executable for the target architecture:

1). Following steps 1 to 9 of the Qt Project Creation and Compilation process for a windowed application, create the project, place the calibration and header files, place the library files for the cross-compilation target architecture (aarch64), create a soft link (symlink) for the dynamic link libraries, create the program in Qt Creator, and select the aarch64 Build Kit

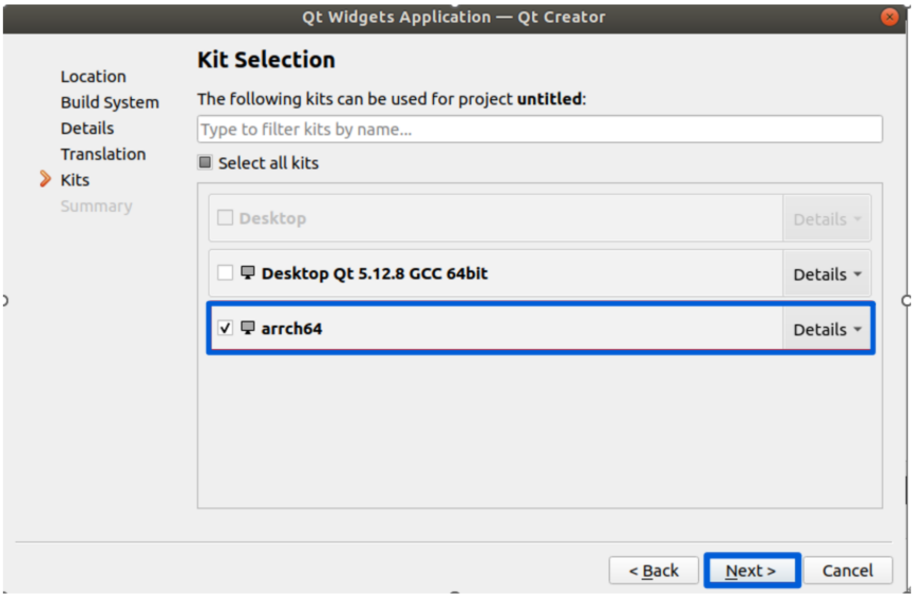

Figure 66 Select the target kit for the cross-compilation target architecture

2). Follow steps 10 to 16 of the windowed application creation process to perform project creation, library referencing, modification of the executable program generation location, project coding, and running;

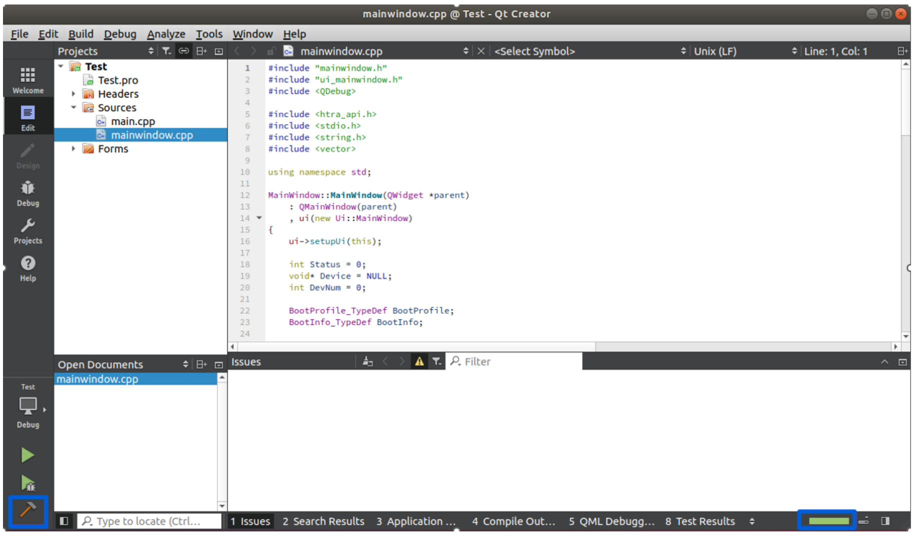

Figure 67 Build the executable program for the aarch64 architecture

3). After building the executable program, open the terminal in the QtTest\bin folder and enter file Test to view the executable program's architecture (the executable program name is Test, so the input command is 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$
```

Figure 68 View the executable program's architecture

2. Run the executable program on the aarch64 architecture host machine:

1). Enter the project directory (The example project is located on the Desktop, so enter cd Desktop/). Execute *zip -r QtTest.zip QtTest* to package the entire QtTest folder into a ZIP file. Then, copy the generated compressed package to the aarch64 host machine;

```
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%)
```

Figure 69 Compress the created project

2). On the aarch64 host machine, enter *unzip QtTest.zip* to unzip (or extract) the project;

3). Refer to the section on Driver File Configuration to configure the driver files on the aarch64 host machine;

4). After configuring the driver files, enter the QtTest folder. In the terminal, enter *chmod 777 Test* to grant permissions to the executable program. Subsequently, you can run the program by entering ./Test.

```
rpdzkj@localhost: ~/Desktop/QtTest/bin
File Edit Tabs Help
rpdzkj@localhost:~/Desktop/QtTest/bin$ chmod 777 Test
rpdzkj@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
```

Figure 70 Grant permissions and run the executable program

8.6 Usage of Python Examples and Project Creation

8.6.1 Usage of Python Examples

Prerequisite: Ensure the device is properly connected and the driver files have been correctly configured according to the Diver File Configuration section.

The functionality of the Python examples can be found in the Python Example Description chapter.

1. Copy the "Linux\HTRA_Python_Examples" folder from the included USB drive to the host PC. In this folder, open a terminal and enter *which python3* to check the location of the Python interpreter (for example, here it is /usr/bin);

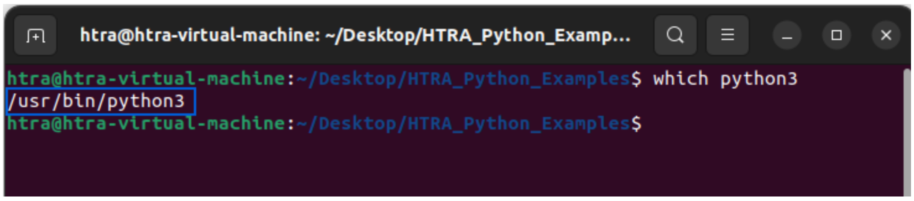

Figure 71 Check the Python3 Interpreter Location

2. Based on the obtained interpreter path, enter *sudo cp -r CalFile /usr/bin* to copy the device calibration files to the same directory as the Python interpreter. (If the interpreter is not located in /usr/bin, replace the path with the actual location.).

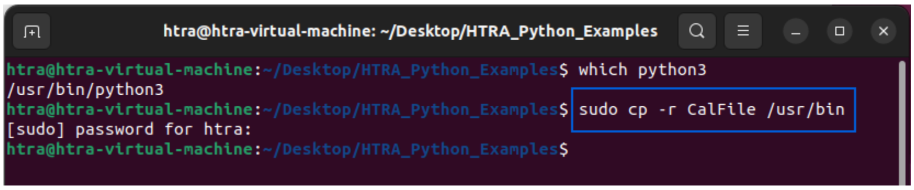

Figure 72 Copy the Calibration Files to the Same Directory as the Interpreter

3. Refer to the Environment Version Compatibility Self-Check chapter to confirm the system architecture, then copy the contents from the corresponding architecture folder under "Linux\Install_HTRA_SDK\htraapi\lib" on the USB drive into the host PC's "HTRA_Python_Examples\htraapi" folder;

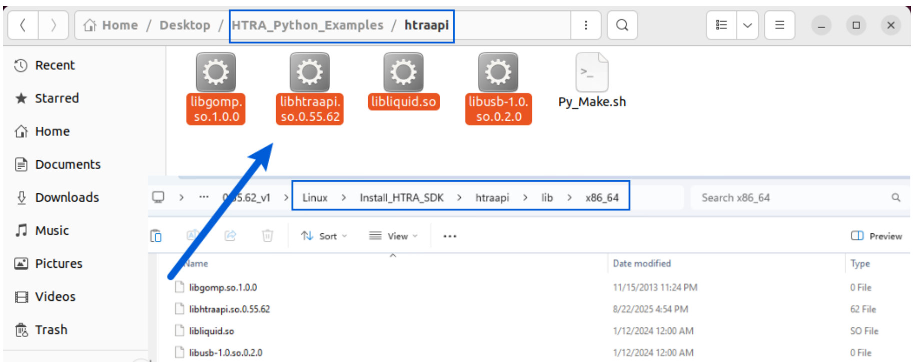

Figure 73 Copy the Dynamic Link Libraries

4. Open a terminal in the htraapi folder, enter *sudo sh Py_Make.sh*, and provide the password when prompted to grant permission for creating symbolic links for the libraries;

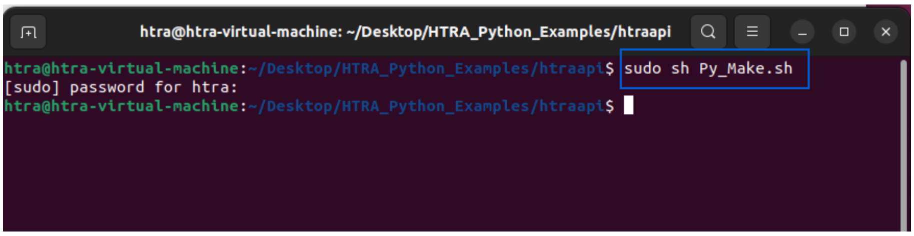

Figure 74 Create symbolic links for the libraries in htraapi

5. In the HTRA_Python_Examples folder, open a terminal and enter *python3 SWPMode_Standard.py* to run the provided SWPMode_Standard.py example.

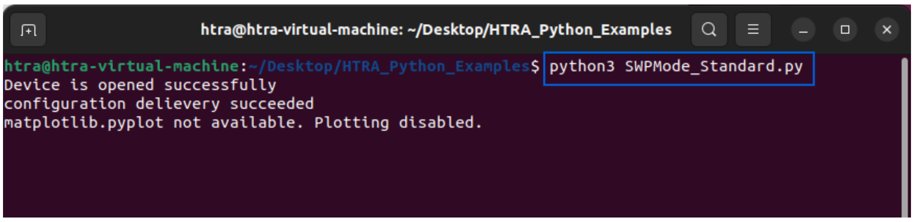

Figure 75 Run the Python Example

Note: If running the Python example on a different device, first copy the calibration files to the "HTRA_Python_Examples\CalFile" folder. Then, in a terminal within the HTRA_Python_Examples folder, enter *sudo cp -r CalFile /usr/bin* to update the calibration files to the same directory as the Python interpreter.

8.6.2 Python Project Creation

Provided that the driver files have been correctly configured as described in the Diver File Configuration section, if you want to create a Python project, follow the procedure below:

1. The code should follow the BNC ICX Series API Programming Guide;
2. When running the program, place it in the example folder (HTRA_Python_Examples) and follow the procedure in the Usage of Python Examples chapter.

Berkeley Nucleonics Corporation

2955 Kerner Blvd.
San Rafael, CA 94901

Phone: (415) 453-9955
Email: info@berkeleynucleonics.com
berkeleynucleonics.com