API 编程指南

API Version 0.54.0

2023-5-17

HAROGIC
海 得 科 技

目录

1 概述

此API系统是基于C编写的动态链接库，用于对设备进行编程开发。

设备的工作模式是API系统的核心概念，不同的工作模式有不同的测试行为与测试能力，开发的第一步就是针对任务要求，选择的合适的工作模式。HTRA API系统的工作模式包括标准频谱分析模式（SWP）、IQ数据记录模式（IQS）、检波分析模式（DET）、实时频谱模式（RTA）。充分理解各工作模式的执行机制，并根据应用目标选择合适的工作模式与设备参数，有助于充分发挥设备的工作效能，并获取更为准确的测量结果。

设备的工作模式与适用场景

标准频谱（SWP）	IQ信号流（IQS）	检波分析（DET）	实时频谱（RTA）
•全景频谱扫描	•时域信号查看	•脉冲信号观察	•突发信号观察
•频谱监测	•IQ记录	•功率时间关系	•隐秘信号发现
•相位噪声	•AM解调		•频谱动态观察
•谐波测试	•FM解调		
•杂散测试	•用户应用		
•信道功率测试			
•OBW、ACPR测试			

API调用的基本流程包括五个步骤：第一步，打开设备资源；第二步，根据工作模式调用相应类别的API函数，将设备配置设备至指定的工作模式与工作参数；第三步，通过该模式下对应的数据获取函数得到测量数据；第四步，利用测量数据，执行用户自定义的分析过程，实现应用目标。第五步，测试结束，关闭设备，释放相关内存资源。

图1　SWP 模式典型的调用步骤

在开始您的应用开发前，请仔细阅读章节——API的调用逻辑与调用地图。以此调用地图作为应用程序的处理框架，有助于快速构建稳健高效的程序。

2 设备及 API 版本

系统是多个软件的联合运行，在本系统中，涉及的软件包括：1）设备主控固件（MFM）、2）FPGA固件（FFM）、3）API、4)SAStudio4等应用程序（如果使用到）。

本系统采用统一的软件版本命名规则来管理所有上述软件。采用x.y.z 的形式命名软件版本。其中x为主版本号，y为次版本号，z为子版本号。次版本y代表了软件的兼容性标识，系统中所有软件必须具有相同的y次版本号，才能严格正确的运行。

例如：MFM版本 = 0.38.1，FFM版本 = 0.38.4，API版本 = 0.38.2，SAStudio4版本 = 0.38.23，此组合所有软件具有y = 38，则可以匹配运行。

例如：MFM版本 = 0.38.1，FFM版本 = 0.38.4，API版本 = 0.54.2，SAStudio4版本 = 0.38.23，此组合中API为超前的版本y = 54，与MFM、FFM、SAStudio4的版本不匹配，则无法运行或可能得到错误的结果。

请注意所使用的API版本与本手册封面所标识的版本对应，以免出现本手册说明与实际API不一致的情况。

3 函数类别简介

表1 API 函数类别简介

函数类别	说明
设备与系统Device	全局功能，可在任意工作模式下调用此类别下的函数。该类别包括对设备打开、关闭、全局性设置、获取设备信息和获取设备状态等功能。
标准频谱分析SWP	该模式下，**接收机根据配置进行跳频以实现频率扫描**，基带对每个频点获取的分析带宽内的时域数据进行**频谱分析**，并将频谱结果返回给用户。SWP模式适用于面向频率迹线的测量与分析应用。 该类别包括对SWP模式进行配置、获取频谱数据、触发控制等功能。
IQ数据记录IQS	该模式下，接收机将中心频点设置为指定频率，并**保持本振频率等接收机状态固定**。基带根据指定的触发信号对分析带宽内的**时域数据进行采集**并返回给用户。IQS模式适用于信号记录、解调分析、同时多维度分析应用。 该类别包括对IQS模式进行配置、获取频谱数据、触发控制等功能。
检波分析DET	该模式下，接收机将中心频点设置为指定频率，并**保持本振频率等接收机状态固定**。基带对分析带宽内的时域信号进行**检波分析**并根据指定的触发信号将功率结果返回给用户。DET模式适用于关注 带内功率-时间 关系的应用，例如脉冲参数测量。 该类别包括对DET模式进行配置、获取频谱数据、触发控制等功能。
实时频谱分析RTA	该模式下，接收机将中心频点设置为指定频率，并**保持本振频率等接收机状态固定**。基带对分析带宽内的时域信号进行连**续频谱分析**，并将频谱结果返回给用户。RTA模式适用于关注瞬时及突发信号的应用，例如干扰排查、复杂电磁环境下特征信号识别等。 该类别包括对RTA模式进行配置、获取频谱数据、触发控制等功能。
多配置调度MPS	该模式下，接收机可下发多个配置，并以时分的方式**按照配置顺序依次获取数据**。 该类别包括MPS模式配置、开始MPS模式、获取数据等功能。
辅助信号源ASG	全局功能，控制设备或其中的**模拟信号源选件，**可在任意工作模式下调用。该类别包括设置输出单音、扫频等功能。
信号处理DSP	通用后处理函数，与硬件状态无关。 该类别包括对IQ数据进行DDC、FFT分析、视频检波等；对频谱谱迹线进行测量分析，比如IM3、相位噪声、信道功率、占用带宽等功能。
模拟调制解调ASD	模拟解调类后处理函数，与硬件状态无关。 该类别包括AM解调、FM解调等功能。

4 Linux 系统中使用 API

4.1 安装配置文件、头文件及 so 库文件

步骤1：确定Linux环境的具体信息。打开终端，通过“uname -a”、“gcc -v”和“ldd --version”三条指令确定当前Linux环境的系统架构、gcc版本及GLIBC版本，如下图所示：

根据终端信息确认当前环境已被支持，若尚未支持，请联系技术支持人员。

表2　Linux 系统下已适配的处理器、编译环境、发行版

X86处理器	**支持intel与AMD处理器**
ARM处理器	arrch64（armv8）、armv7处理器，例如：树莓派4b、RK3399、RK3568、RK3588、T507、NIVIDA Jeston TX2
编译环境	gcc 5.3.1、glib 2.21
发行版	树莓派4b定制系统、ubantu 18.04等

步骤2：确认Linux资料文件夹中的内容，内容包括：

1）configs文件夹内为Linux环境中设备的驱动文件；

2）example文件夹内为一个范例可执行文件和CalFile文件夹；

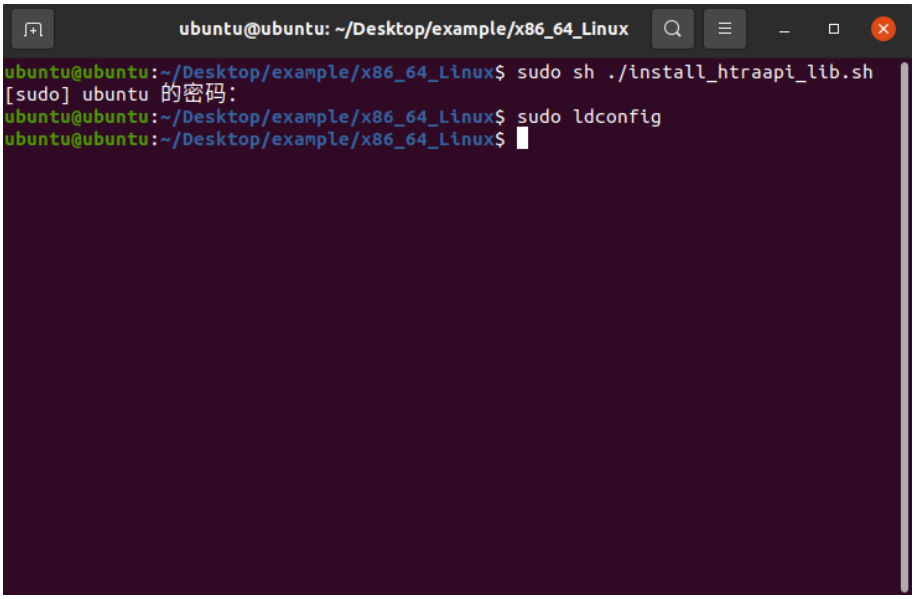
3）inc文件夹内为htra_api头文件；

4）lib文件夹内为编译的so文件；

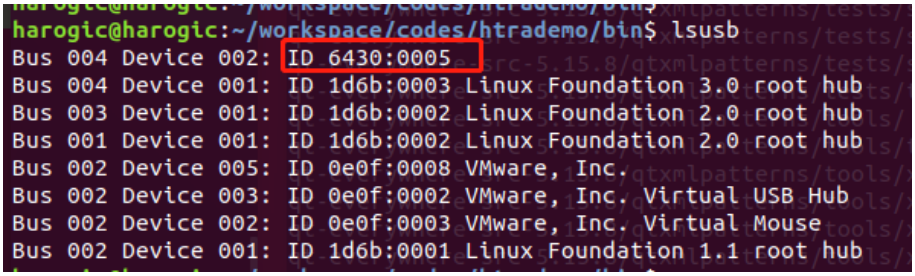
5）sh文件为配置脚本，使用前请先阅读README文件。具体内容如下图所示：

若未找到对应环境的资料，请联系技术支持人员获取。

步骤3：通过指令“sudo sh ./install_htraapi_lib.sh”运行.sh文件，安装需要的配置文件、.so库文件和.h库文件。注意：有些版本的Linux系统在安装完新库之后需要声明一下，通过“sudo ldconfig”指令，使系统正确知道已经安装了新的lib库。如下图所示：

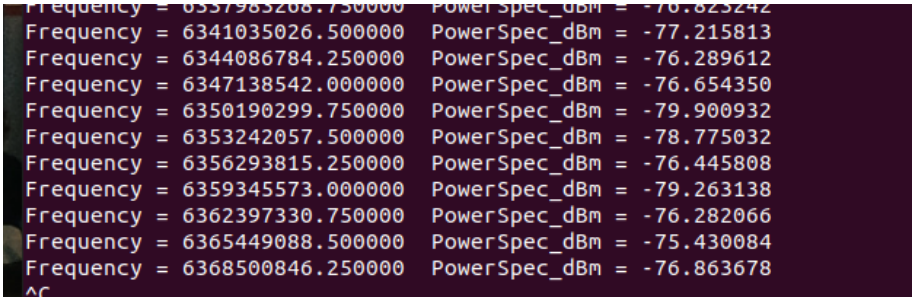


步骤4：在安装完成驱动及库后，接上设备，可先通过指令“lsusb”确定Linux环境下是否有设备接入，如下图所示，其中“ID：6430”即为接入设备。



步骤 5：运行已经编译好的范例可执行文件。进入 example 文件夹（注意：需要将设备的校准文件拷贝至 CalFile 文件夹，CalFile 文件夹需要与可执行程序在同一级目录），在终端输入“ chmod 777 SWPMode_Standard ”，再在终端输入“ ./SWPMode_Standard ”运行

SWPMode_Standard 范例，如果正常运行会有频谱打印信息。如下图所示：

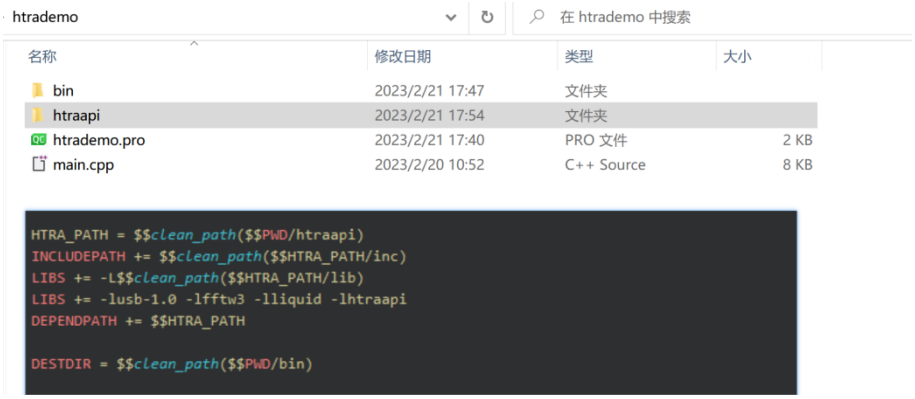


步骤 6：实际调用 htra_api。参考 example 文件夹，将发货资料中的 CalFile 文件夹和范例拷贝到新建文件夹中，例如将 SWPMode_Standard.cpp 文件拷贝过来。然后通过指令：“g++ -o SWPMode_Standard SWPMode_Standard.cpp -std=c++11 -lhtraapi -lliquid -lfftw3 -lusb-1.0 -lpthread”进行编译。编译完成后即可在终端中输入./SWPMode_Standard，运行可执行程序。

4.2 在 QT 中使用 API 与范例

步骤 1：创建 Qt 工程。

步骤 2：在.pro 文件中配置.so 库文件和.h 库文件，配置如下图所示，其中 htraapi 文件夹为.so 及头文件。



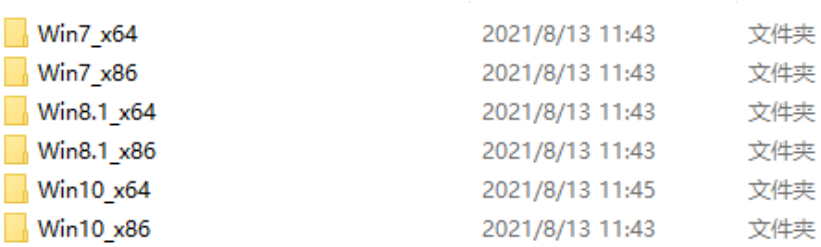
步骤 3：在 bin 文件夹中将 CalFile 文件夹放置在内，编译出的可执行文件目标路径（CalFile 文件夹与可执行程序必须在同一级目录）。

步骤 4：执行 qmake，然后重新构建即可正常调用库文件。

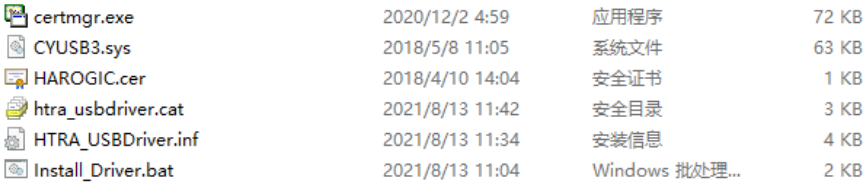
5 Windows 系统中使用 API

5.1 安装设备驱动程序

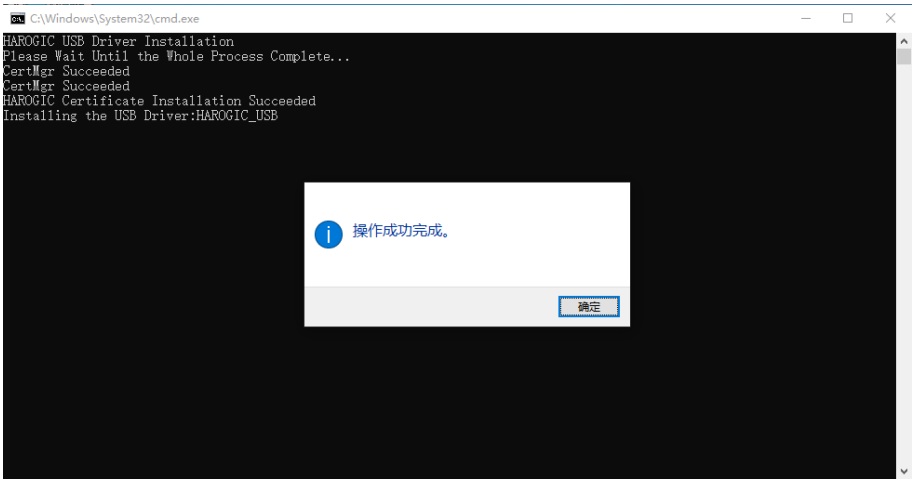
（1）安装驱动前，请确认Windows版本，然后选择您电脑对应的驱动版本进行安装，如下图所示。



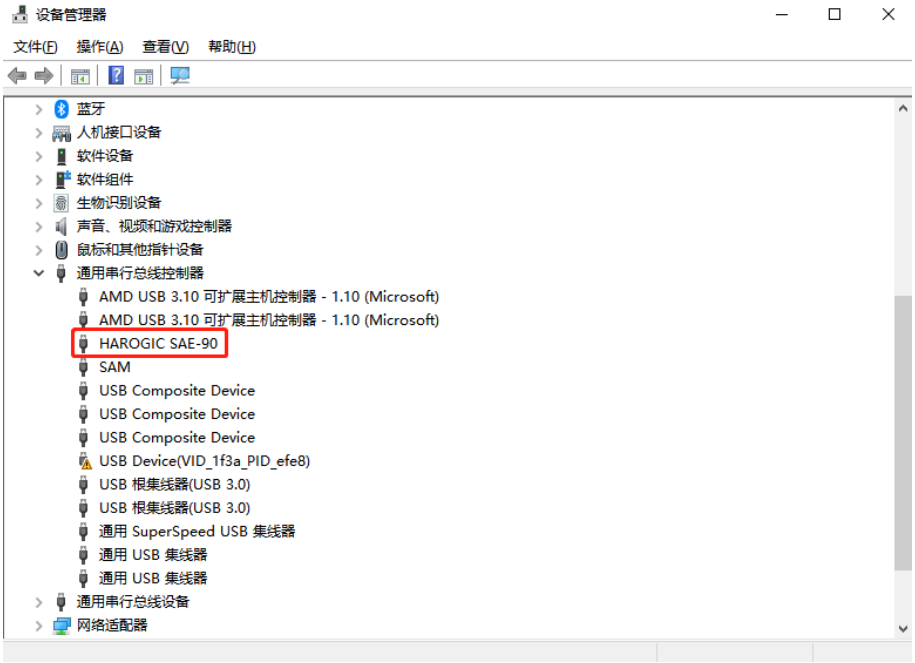
（2）以管理员身份运行Install_Driver.bat文件，如下图所示。



（3）驱动安装成功后，如下图所示。

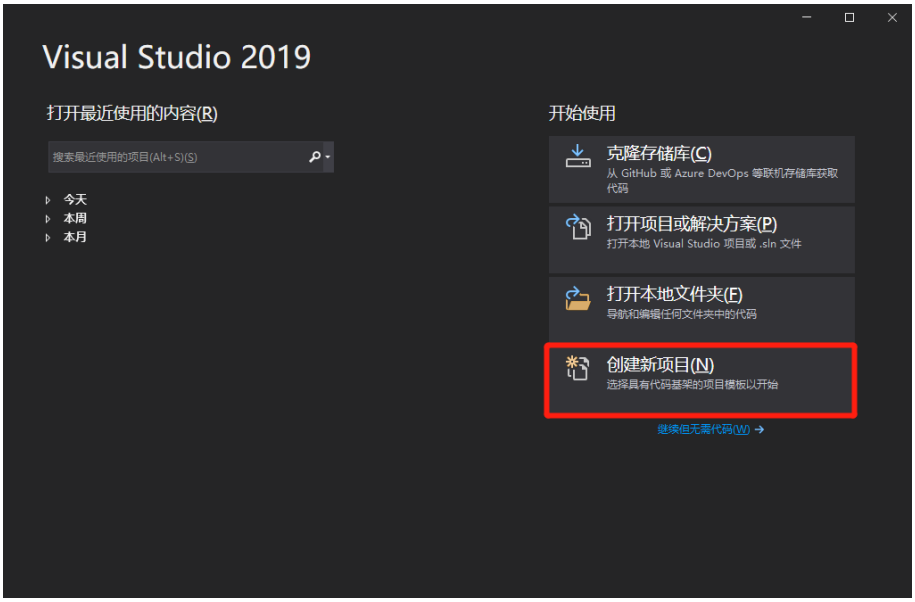


（4）成功安装后可在设备管理器中查看新安装设备，如下图所示：

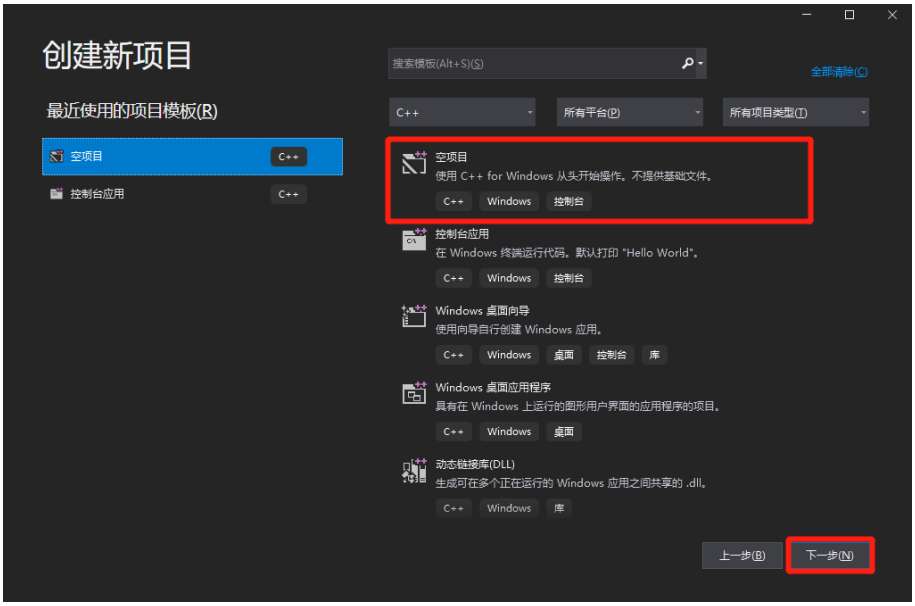


5.2 在 C 开发环境中使用 API（VS2019）

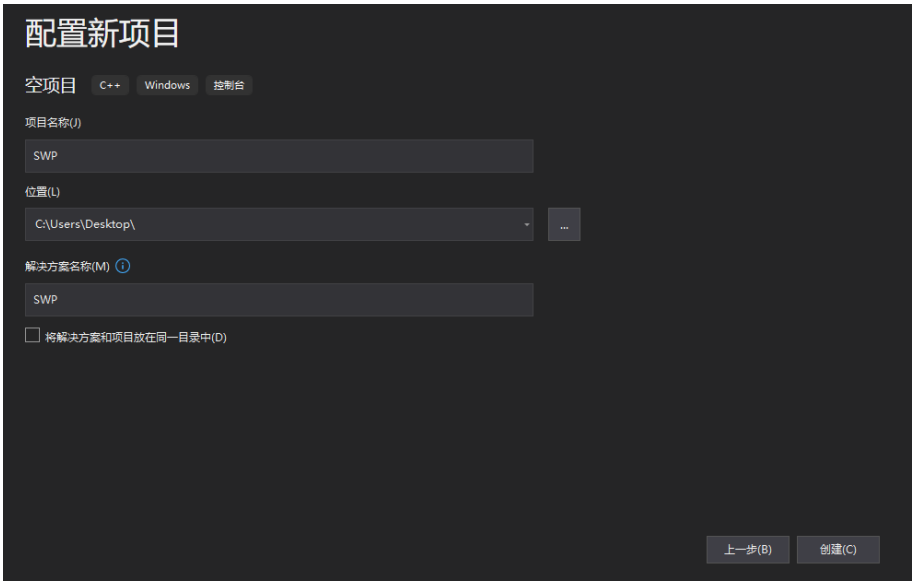
步骤1：打开Visual Studio 2019，点击创建新项目，如下图所示。



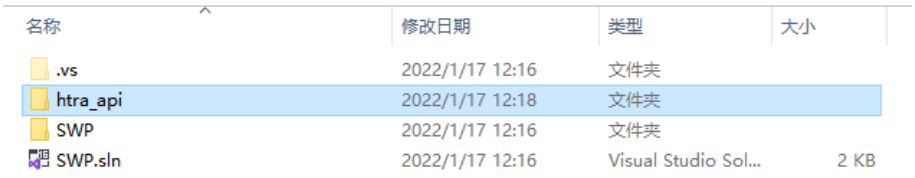
步骤2：选择空项目，并点击下一步，如下图所示。



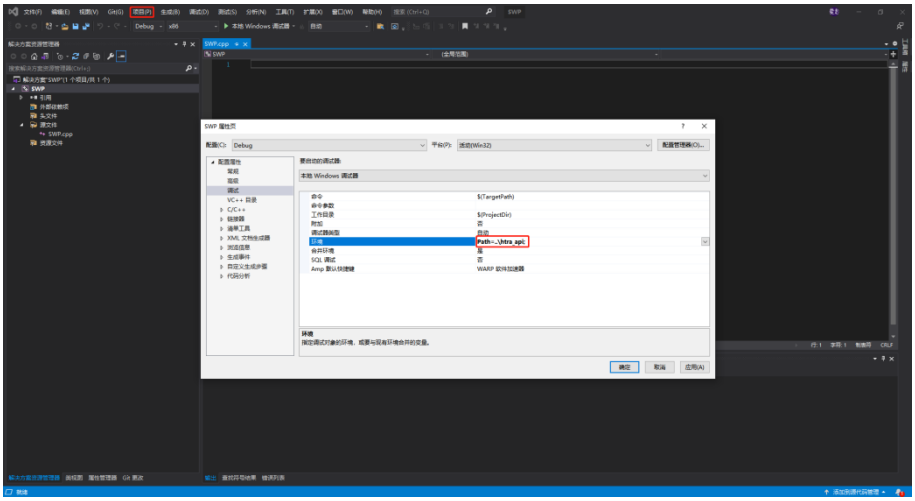
步骤3：将项目名称命名为需要的名字，例如SWP；项目位置设置为需要存放的位置，例如桌面；解决方案名称可与项目名称同名。将解方案和项目放在同一目录中，可以不勾选；最后点击创建，如下图所示。



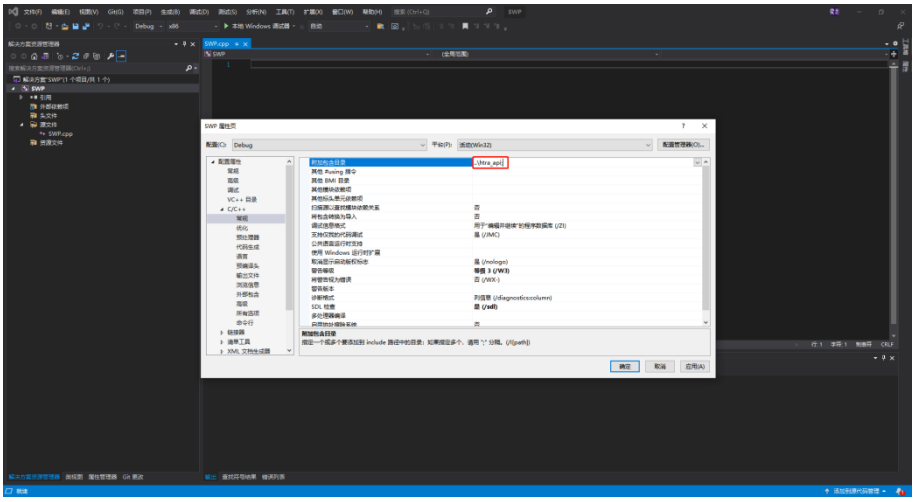
步骤4：创建完成后，将发货u盘中的\Windows_API\x86\htra_api文件夹放在工程平级目录下，如下图所示。



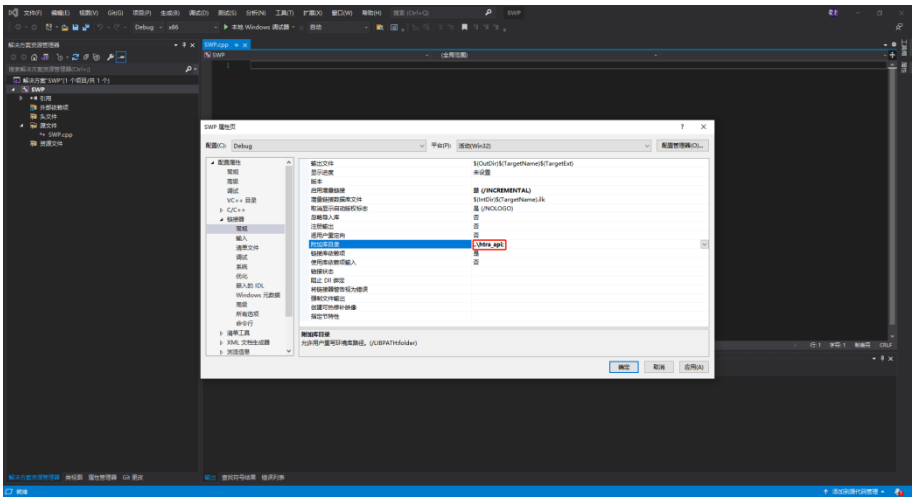
步骤5：双击打开SWP.sln，在源文件中新建一个SWP.cpp文件，然后点击上方菜单栏中的项目 > 属性，将配置属性 > 调试中的环境设置为Path=..\htra_api，如下图所示。



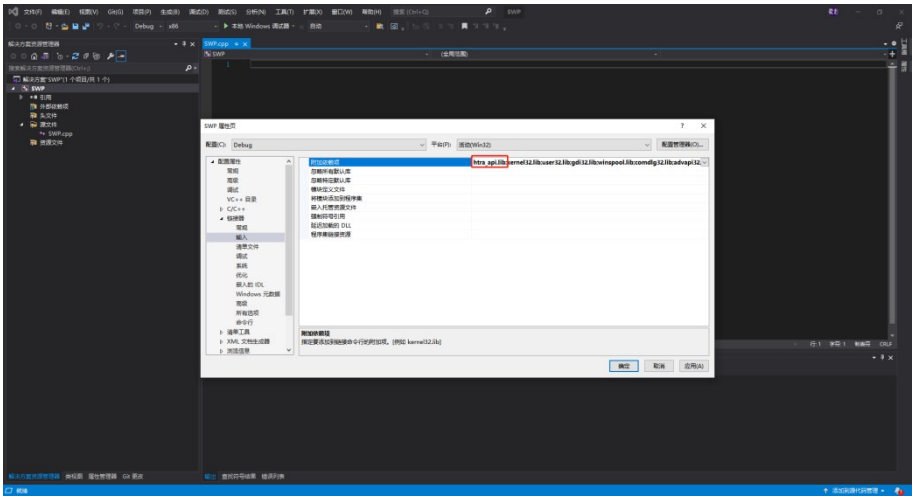
步骤6：将配置属性 > C/C++ >常规中的附加包含目录设置为..\htra_api，如下图所示。



步骤7：将配置属性 > 链接器 >常规中的附加库目录设置为..\htra_api，如下图所示。



步骤8：给配置属性 > 链接器 > 输入中的附加依赖项新增htra_api.lib，如下图所示。



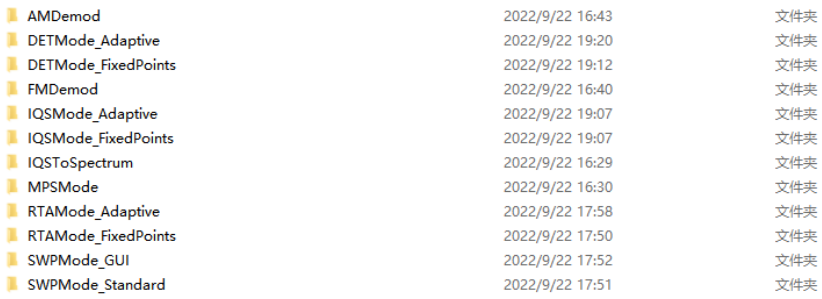
至此，C/C++工程配置结束，可以进行编程开发。

5.3 API 的 C 范例

提供了4个模式下的多个范例，每个范例有单独的工程，可独立使用。

表3　四个模式范例说明

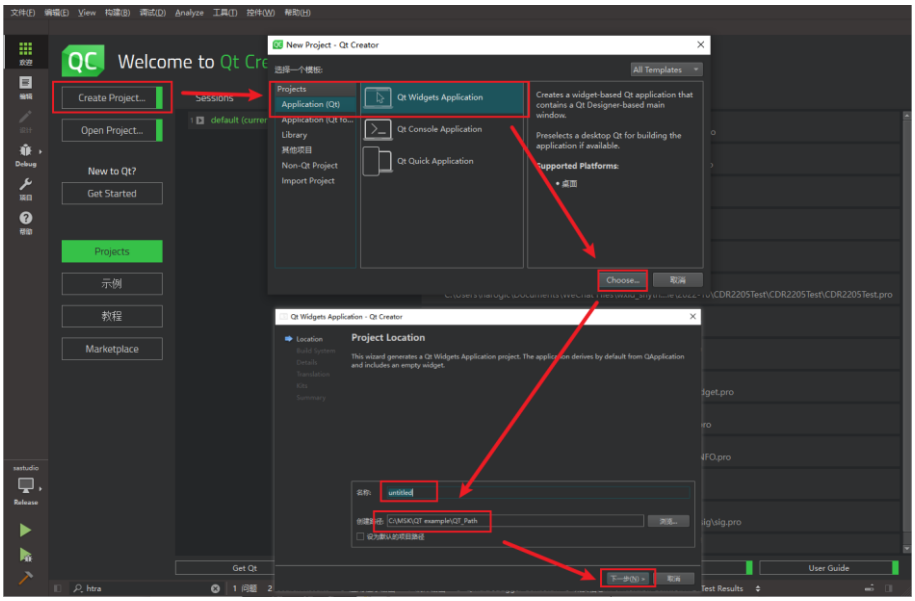
序	范例	说明
1	SWPMode_Standard	标准频谱仪模式，配置并获取频谱。
2	SWPMode_GUI	调用API开发GUI应用的参考范例。
3	IQSMode_FixedPoints	IQS定点模式，配置并获取用户指定长度的IQ数据
4	IQSMode_Adaptive	IQS触发模式，用户配置触发开始及结束的信号，并获取在触发信号有效期间的IQ数据。
5	DETMode_FixedPoints	DET定点模式，配置并获取用户指定长度的DET数据。
6	DETMode_Adaptive	DET触发模式，用户配置触发开始及结束的信号，并获取在触发信号有效期间的DET数据。
7	RTAMode_FixedPoints	RTA定点模式，配置并获取用户指定长度的RTA数据。
8	RTAMode_Adaptive	RTA触发模式，用户配置触发开始及结束的信号，并获取在触发信号有效期间的RTA数据。
9	IQSToSpectrum	IQ数据转换为频谱数据范例。
10	MPSMode	多配置模式获取数据的范例。
11	FMDemod	对IQ数据进行FM解调的范例。
12	AMDemod	对IQ数据进行AM解调的范例。

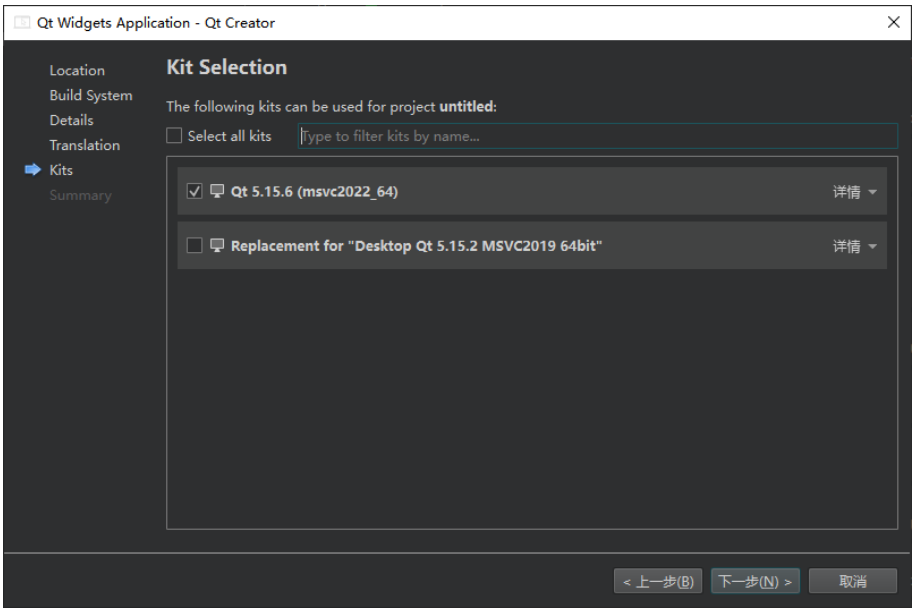


以SWPMode_Standard工程为例，htra_api文件夹下包含了频谱仪相关的所有文件：CalFile文件夹及.dll、.lib、.h文件。CalFile文件夹中包含适配本机的校准文件。用户需要将htra_api文件放在工程平级目录，方可使用例程，如下图所示。

5.4 在 QT 中使用 API 与范例

步骤1：打开Qt Creator，创建工程，选择合适命名及创建路径，如下图所示；选择合适基类创建，然后选择对应Kits套件，然后创建完成如下图所示：

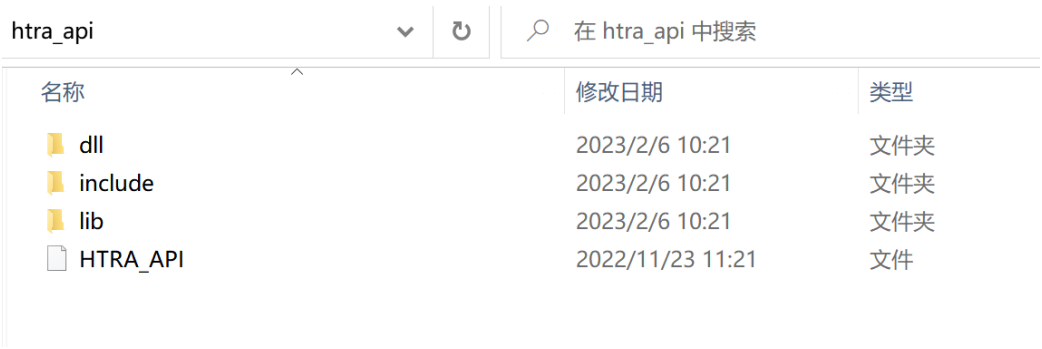




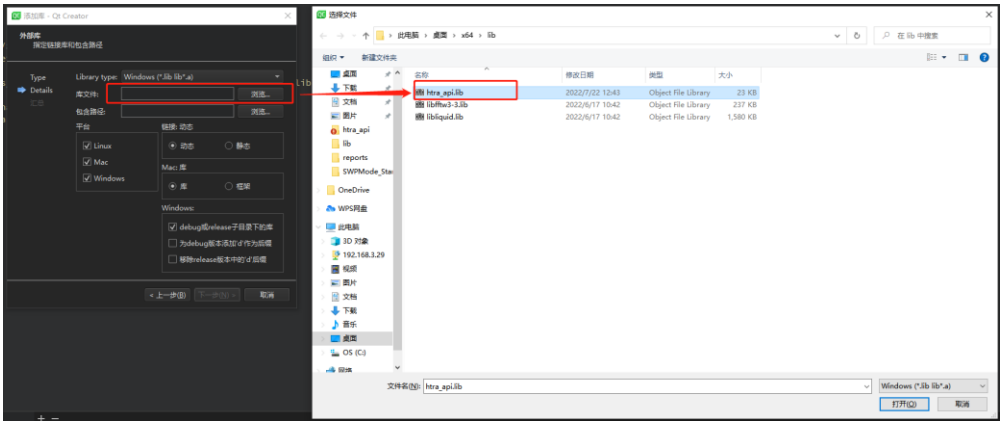
步骤 2：在 Qt 工程中添加 htra_api 及三方库。以添加 x64 位为例，具体流程如下所示：

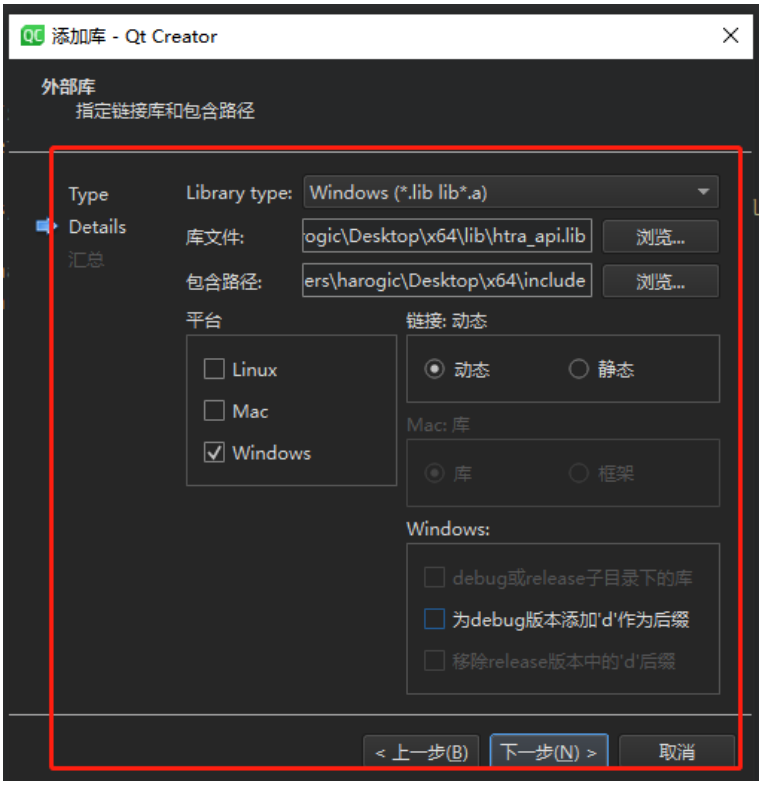
1）新建一个文件夹如 htra_api，建立三个文件夹分别为：dll、lib、include；

2）将 U 盘中 x64 库文件夹中对应后缀文件放置到对应文件夹内，其中三个 dll 和 CalFile 文件夹都放在 dll 文件夹。如下图所示；

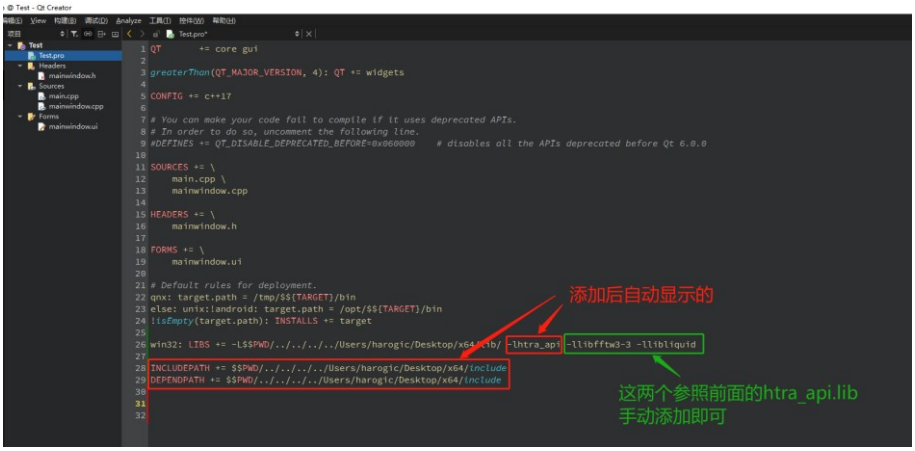


3）在 Qt 中添加库 > 外部库 > 库文件 > 选择 lib 库文件，选中一个即可，点击打开，如下图所示（将 htra_api 文件夹放置在 Qt 工程路径中）；



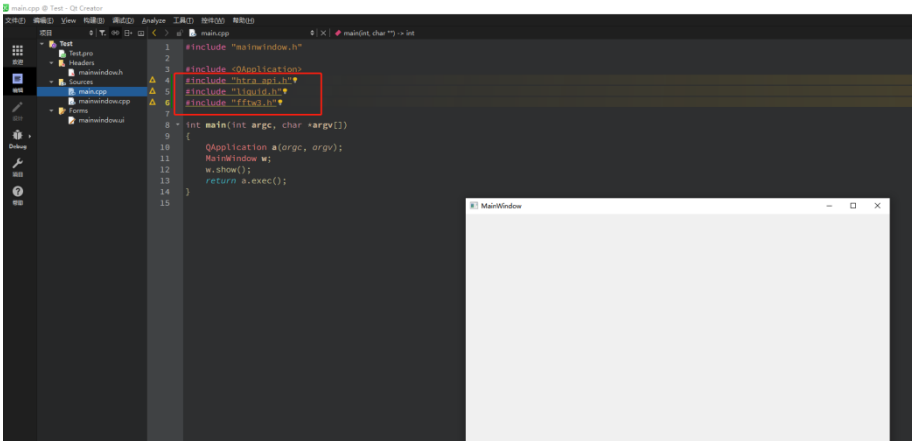


4）点击下一步完成，然后进入到.pro 文件内，将余下 libfftw3-3.lib 和 ibliquid.lib 参照已添加的 htra_api.lib 添加即可，添加完成后如下图所示；



5）执行 qmake，然后重新构建。将...\htra_api\dll 文件夹中的 CalFile 文件夹、htra_api.dll、libfftw3-3.dll、libliquid.dll 复制粘贴到.exe 的同路径下，如下图所示：

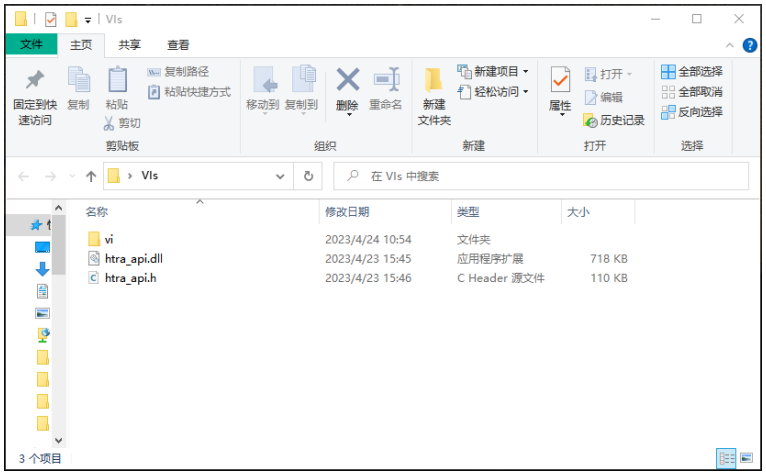
步骤 3：此时库添加完成，可以正常调用库文件，如下图所示：



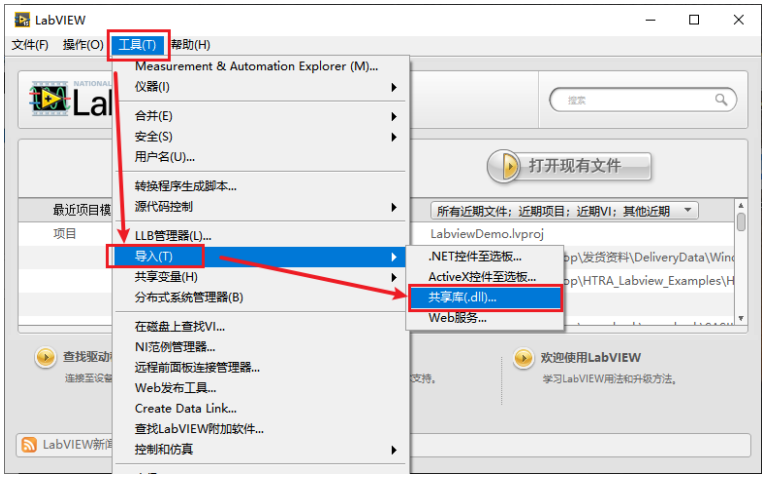
5.5 在 Labview 中使用 API 与范例

首先，需要再Labview环境中（以Labview 2015为例）导入htra_api.dll相关函数

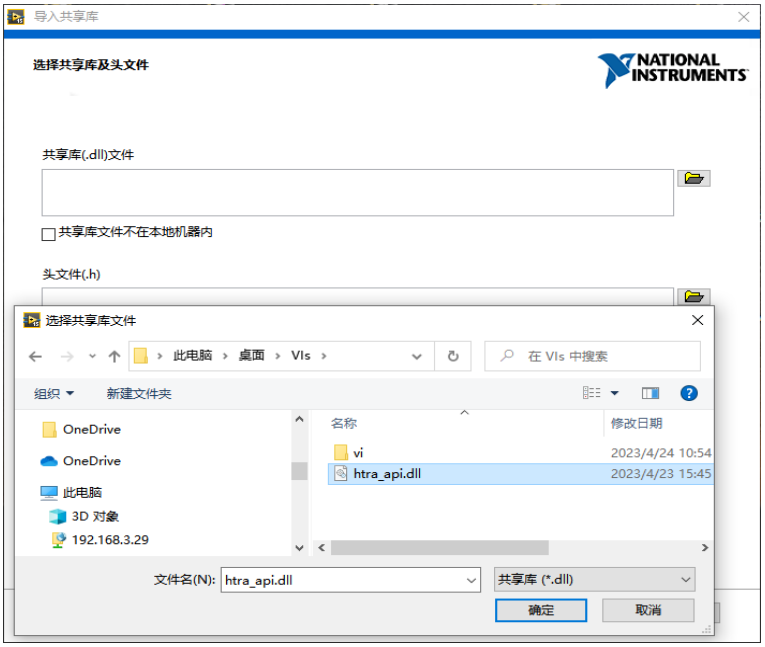
步骤1：创建一个和文件夹，将U盘中Windows_API\x86内htra_api.dll和htra_api.h拷贝该文件夹中，再创建出一个文件夹用于放置htra_api.dll导出的函数（注意：需要将htra_api.h编码格式改为utf-8，且将需要导出函数中参数类型为uint64_t改为double型，导出函数后在vi中将参数类型改回uint64_t型），如下图所示：



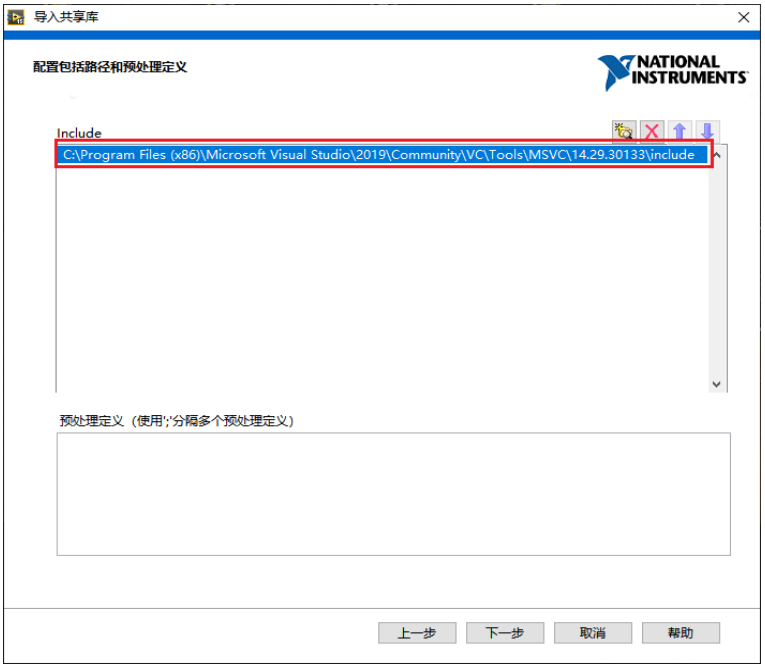
步骤2：打开Labview，导出API函数。如下图所示：



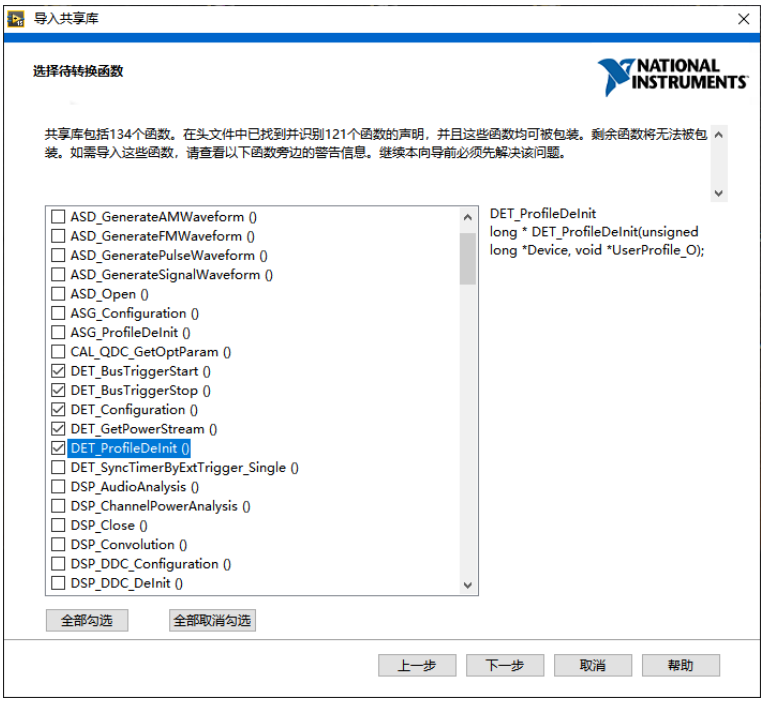
步骤3：新界面中选择为共享库创建VI，点击下一步，共享库及头文件路径选择刚创建好的文件夹中对应文件，选择好共享库路径后，头文件路径可以自动识别。如下图所示：



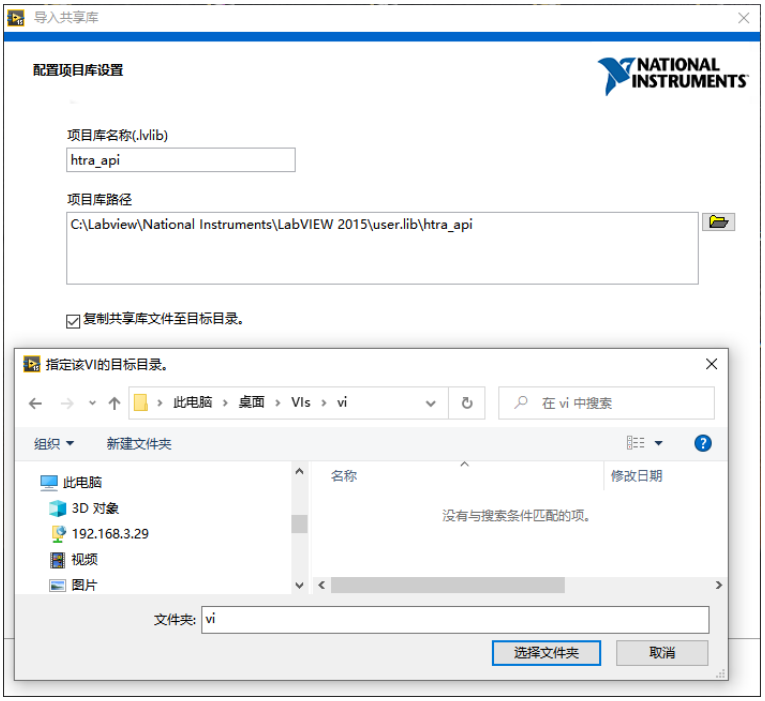
步骤4：点击下一步，配置包括路径与处理定义，预处理定义可以不做选择。如下图所示：



步骤5：点击下一步，选择需要导出的函数（建议每次导出一个模式）。如下图所示：

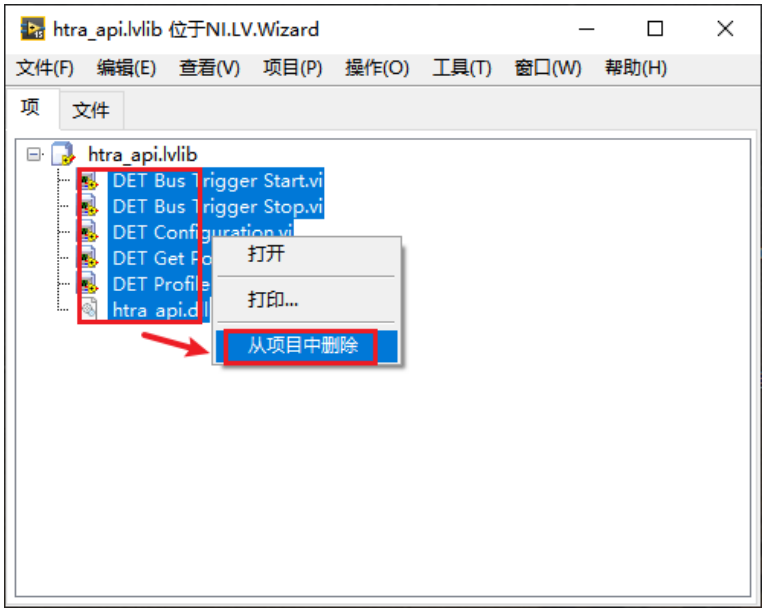


步骤6：点击下一步，项目库路径选择存放共享库和头文件路径中的vi文件夹即可，其它配置选择默认即可，然后错误处理方式选择建议错误处理，点击下一步，如下图所示：

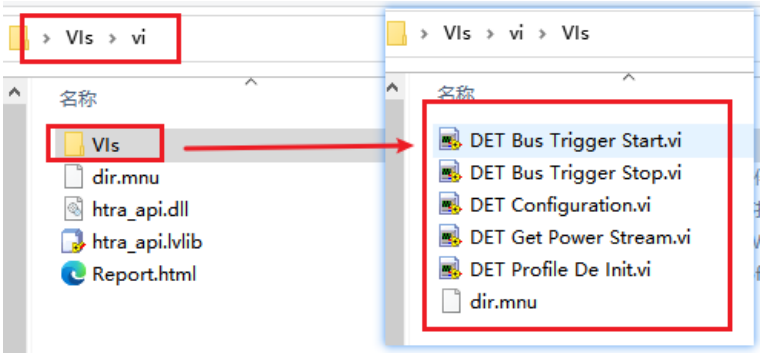


步骤7：函数调用库节点设置在任意线程中运行，下一步，等待生成函数。如下图所示：

步骤8：勾选打开生成库，查看报告可不勾选，点击完成，在弹出的htra_api.lvlib中将导出的函数及dll从项目中删除，删除后关闭htra_api.lvlib。如下图所示：



步骤9：导出完成后文件夹内为API函数。如下图所示：

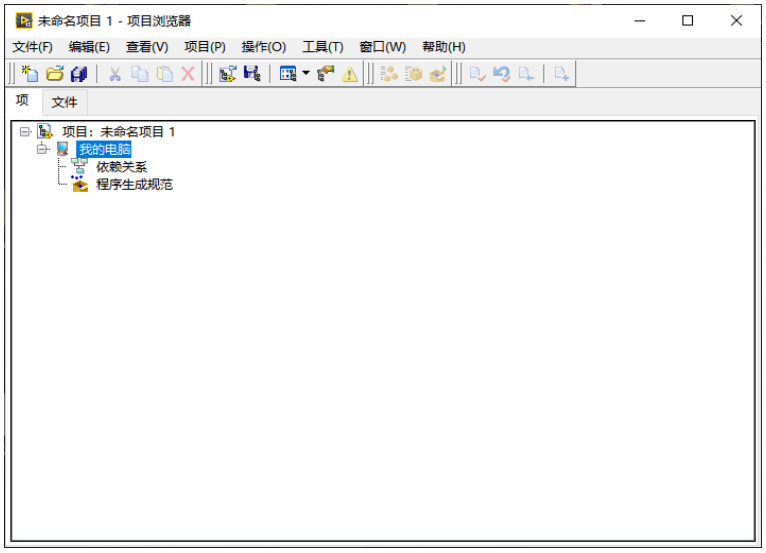


至此，Labview环境中导出API函数结束。

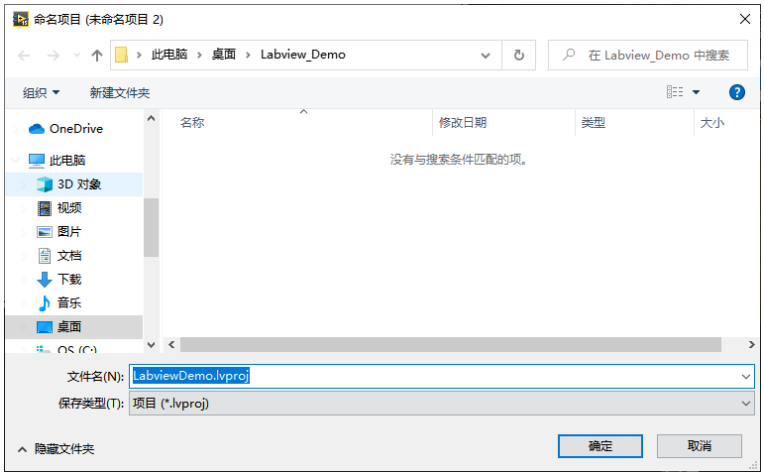
进一步在Labview环境中调用htra_api.dll的相关函数

步骤1：打开Labview，点击创建项目，如下图所示：

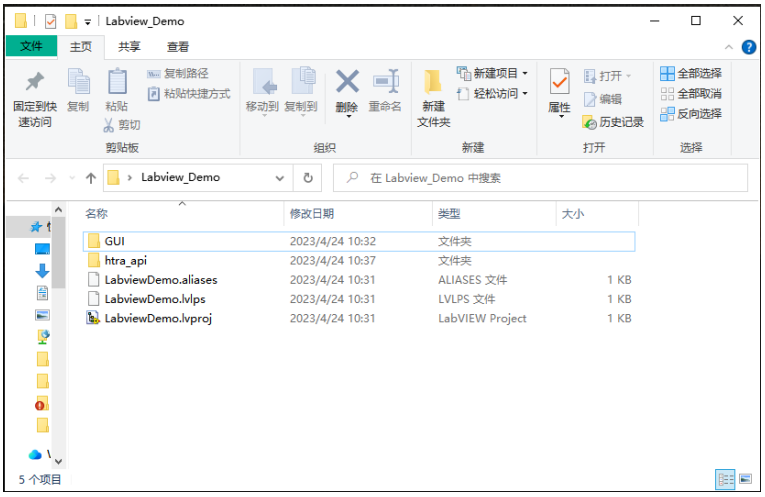
步骤2：点击完成，此时会创建出一个未命名的空项目，如下图所示：



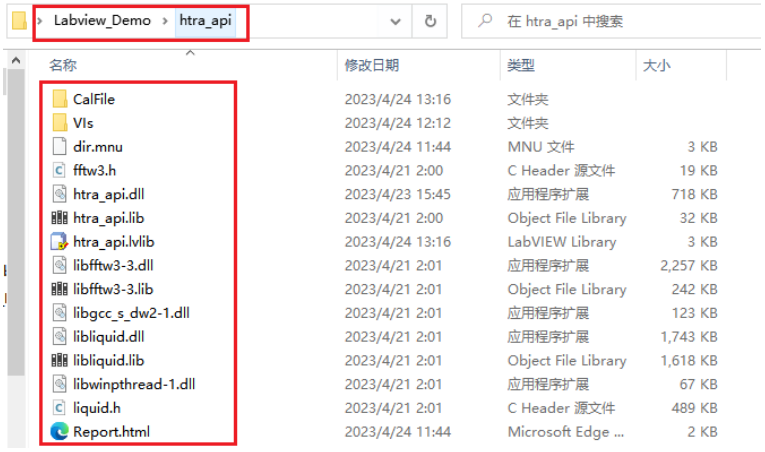
步骤3：保存项目，选定预保存路径，并给项目命名，然后确定，如下图所示：



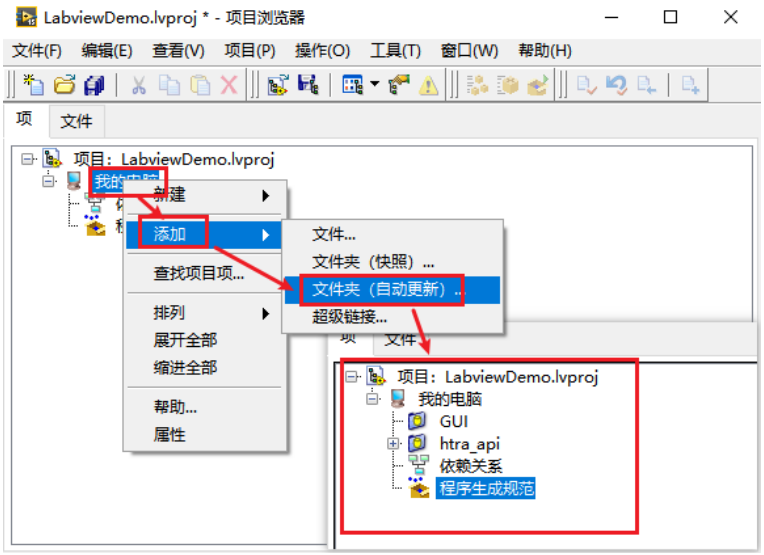
步骤4：在项目同级目录下创建三个文件夹，例如命名为GUI，用于存放范例；再创建一个文件夹如htra_api，用于存放htra_api.dll库、CalFile文件夹以及htra_api.dll导出的Labview函数；如果有需要可以再创建一个文件夹如Subvi，用于存放子vi。如下图所示：



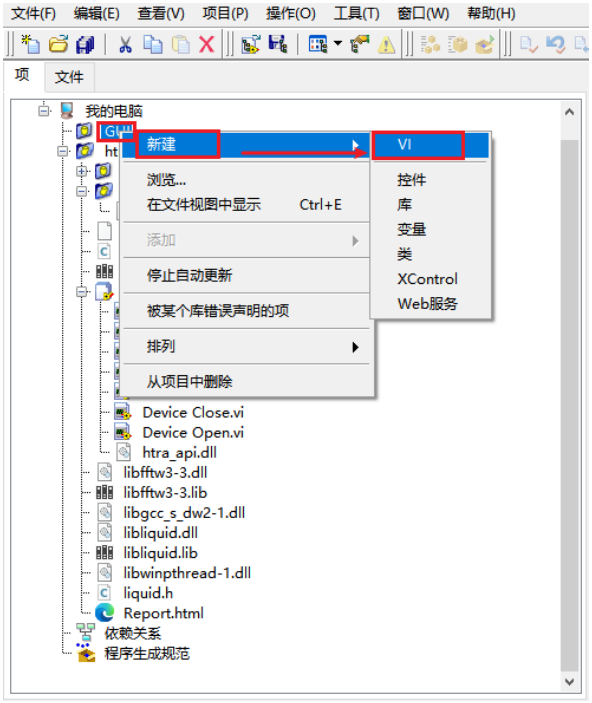
步骤5：打开vi文件夹，将文件夹内容拷贝至创建的Labview项目中htra_api文件夹；并将U盘中Windows_API\x86文件夹下所有内容也拷贝至Labview项目中htra_api文件夹中。如下图所示：



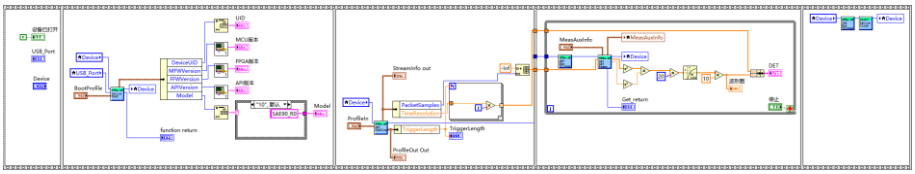
步骤6：打开工程，将已配置好的GUI文件夹和htra_api文件夹添加至工程中，然后保存：



步骤7：在GUI文件夹中创建VI，例如命名为DET_Example。如下图所示：

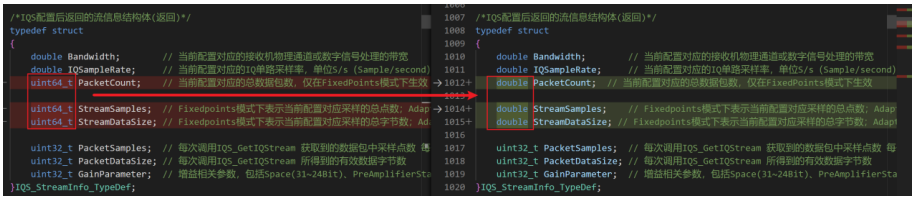


步骤8：在程序框图页内即可调用导出的Labview API函数，调用流程与C环境中一致。如下图所示：

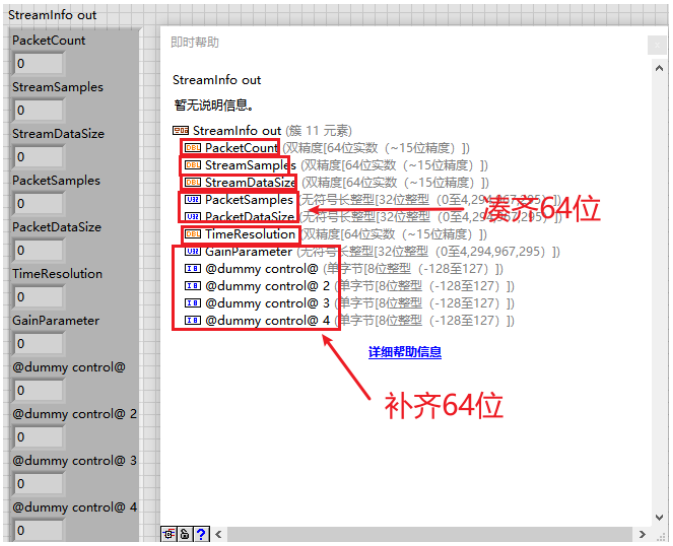


至此，Labview环境中调用API函数结束。

注意：由于Labview导出dll库时，无法识别uint64_t类型参数，所以需要将导出函数中参数类型为uint64_t改为double型，成功导出函数后需要在vi中将参数类型改回uint64_t型；也可以直接在提供的Labview范例基础上做修改。如下图所示：



导出函数之后，请格外留意结构体中参数位数大小，Labview导出的结构体需要做补位到64位，否则，实际参数有可能会串位导致显示读取有误，如下图所示：



5.6 在 Matlab 中使用 API 与范例

以下以如何调用64位htra_api为例说明，32位的调用方法与64位基本相同。

步骤1：安装MSYS2和mingw-w64 GCC，下载和安装链接：**https://www.msys2.org/**。

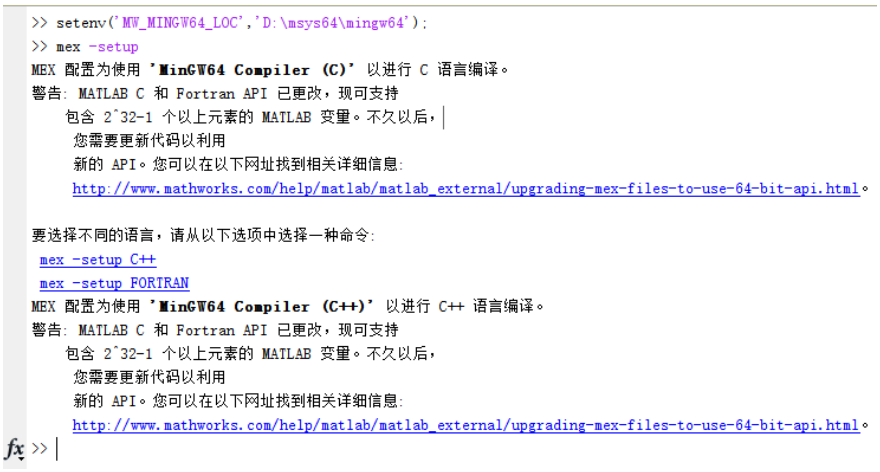
步骤2：配置Matlab

（1）在Matlab终端输入：**setenv('MW_MINGW64_LOC','D:\msys64\mingw64');**

注意：D:\msys64\mingw64 地址为你电脑上安装mingw64.exe的地址

（2）在Matlab终端输入：**mex -setup**，到此即完成C语言编译器的配置

（3）如需使用C++编译器，则鼠标左击终端中的 mex -setup C++



步骤3：调用dll

（1）加载dll：loadlibrary('.\htra_api\htra_api.dll', '.\htra_api\htra_api.h');　　%.dll和.h的文件路径一定要注意，.\ 代表当前.m文件路径。

（2）判断dll是否已加载：libisloaded('htra_api.dll');

（3）指针和结构体指针。对于Matlab来说，没有C语言中的类似指针的变量。可以通过 libpointer 函数创建变量指针，然后通过get()函数来获取指针指向的值。可以通过 libstruct 函数创建结构体指针，然后通过get()函数来获取结构体指针指向的值。可以将 libstruct 函数理解为将Matlab的结构体转为C语言的结构体。

通过 libpointer 或 libstruct 创建的变量，用与后续的使用。若不使用libpointer或libstruct是无法正常调用C的函数或无法获取到值。

```
BootInfo.DeviceInfo = DeviceInfo;
BootInfo.BusSpeed = 0;
BootInfo.BusVersion = 0;
BootInfo.APIVersion = 0;
BootInfo.ErrorCodes = 1:7;
BootInfo.Errors = 0;
BootInfo.WarningCodes = 1:7;
BootInfo.Warnings = 0;

BootInfo_p = libstruct('BootInfo_TypeDef',BootInfo);

Status = calllib('htra_api', 'Device_Open', Device, DevNum,BootProfile_p,BootInfo_p);

get(BootInfo_p);
```

```
IQ_data = int16(1:StreamInfo_p.PacketSamples * 2);
IQ_data_p = libpointer('int16Ptr',IQ_data);

I_data = 1:StreamInfo_p.PacketSamples;
Q_data = 1:StreamInfo_p.PacketSamples;

for t=1:30

    Status = calllib('htra_api', 'IQS_BusTriggerStart', Device);
    Status = calllib('htra_api', 'IQS_GetIQStream_Data', Device, IQ_data_p);

    for i=1:StreamInfo_p.PacketSamples
        I_data(i)=IQ_data_p.value(2 * i - 1);
        Q_data(i)=IQ_data_p.value(2 * i);
        %fprintf('%d\n',i)
    end

    % figure('Name' , 'IQ');
    plot(1:StreamInfo_p.PacketSamples,I_data,1:StreamInfo_p.PacketSamples,Q_data);
    axis([0, StreamInfo_p.PacketSamples - 1, -50, 50]);
     pause(0.000001);

end
```

（4）定义指针变量的时候，注意一点要定义成*Ptr，例如uint32Ptr。

```
a=0;
b=0;
c =0;
SamplePoints = libpointer('uint32Ptr',a);
DataByte = libpointer('uint16Ptr',b);
IQSDataStartIndex = libpointer('uint32Ptr',c);


FilePath='.\20230306_194831.iq.wav';
Status = calllib('htra_api', 'Device_GetIQSDataInfo', FilePath, SamplePoints, DataByte, IQSDataStartIndex);
```

6 API 的调用逻辑与调用地图

HTRA API编程的核心步骤包括：打开设备、配置设备、获取数据、错误与警告处理。

6.1 标准扫频分析（SWP）的 API 调用地图

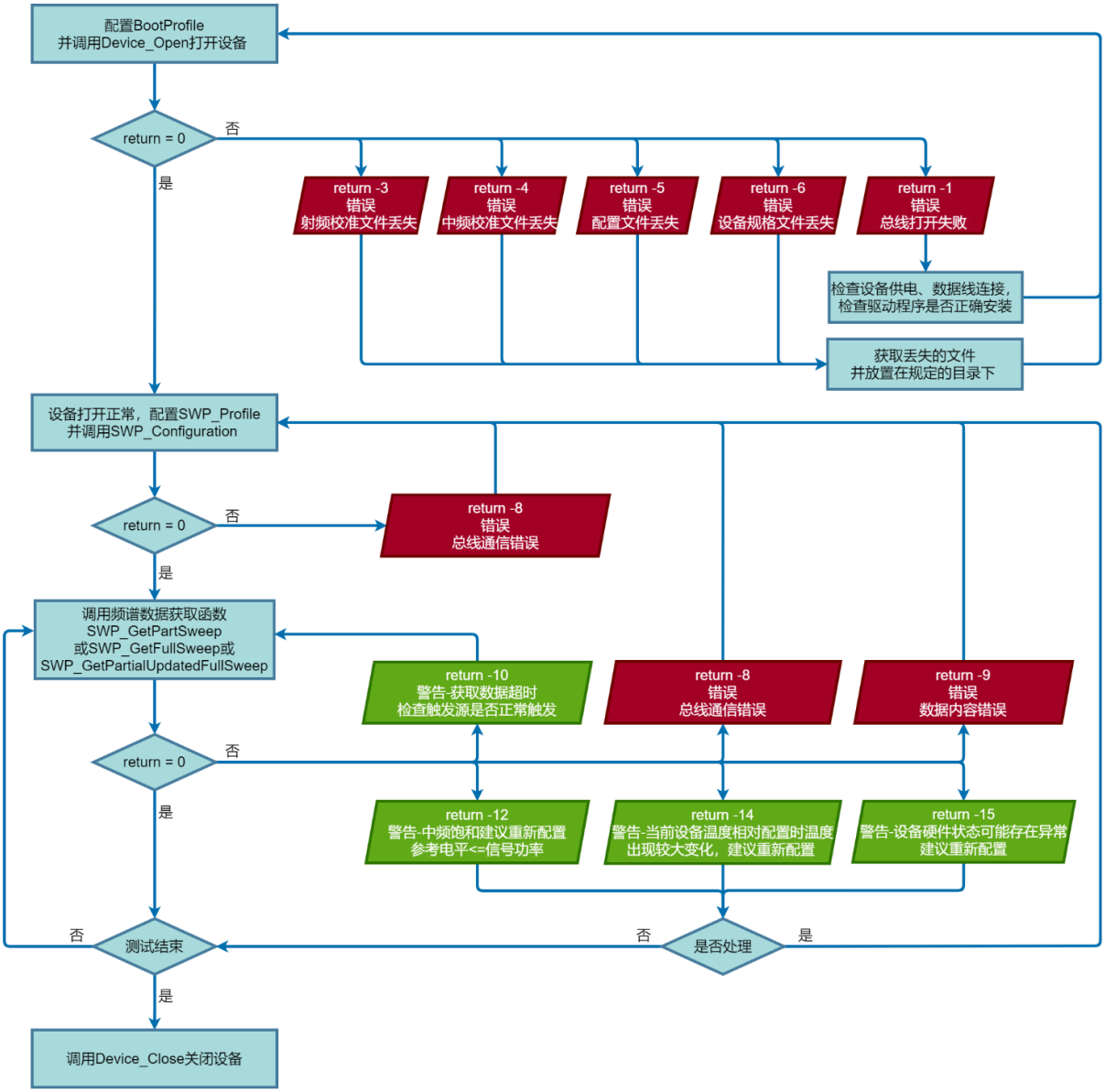


图2　1SWP 模式调用流程图

6.2 时域信号记录分析（IQS）的 API 调用地图

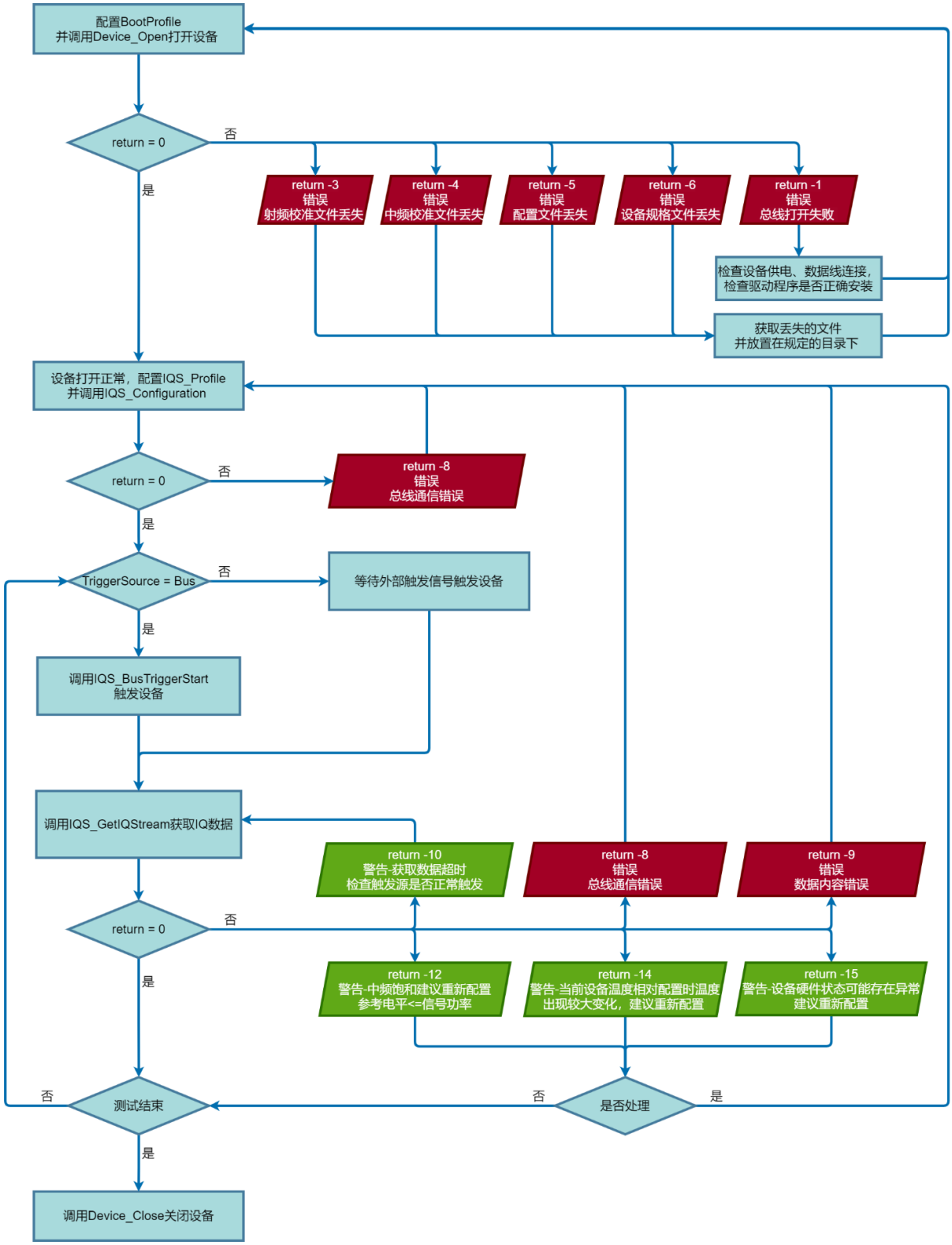


图3　IQS 模式调用流程图(触发模式为 Fixed)

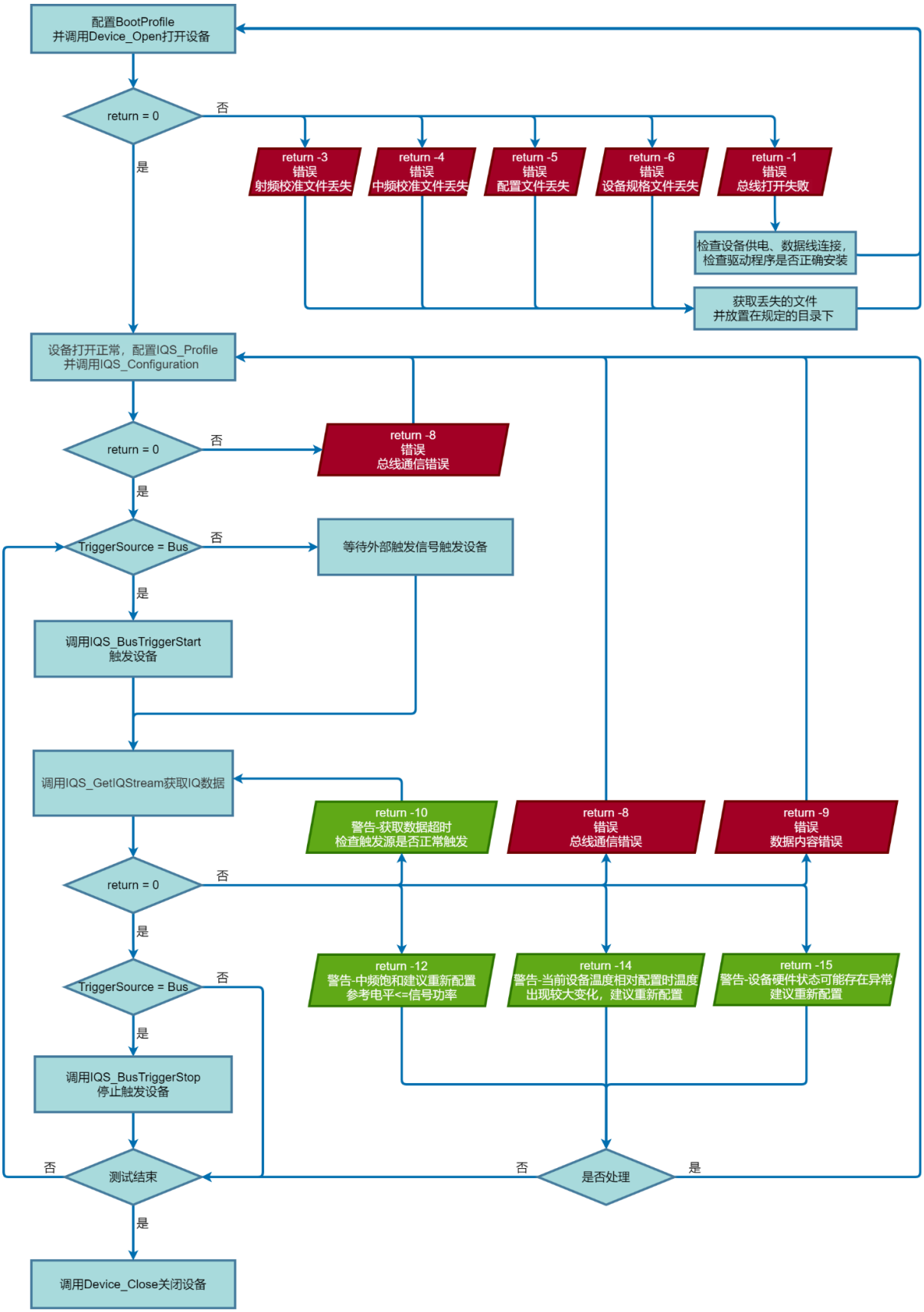


图4　IQS 模式调用流程图(触发模式为 Adaptive)

6.3 检波分析（DET）的 API 调用地图

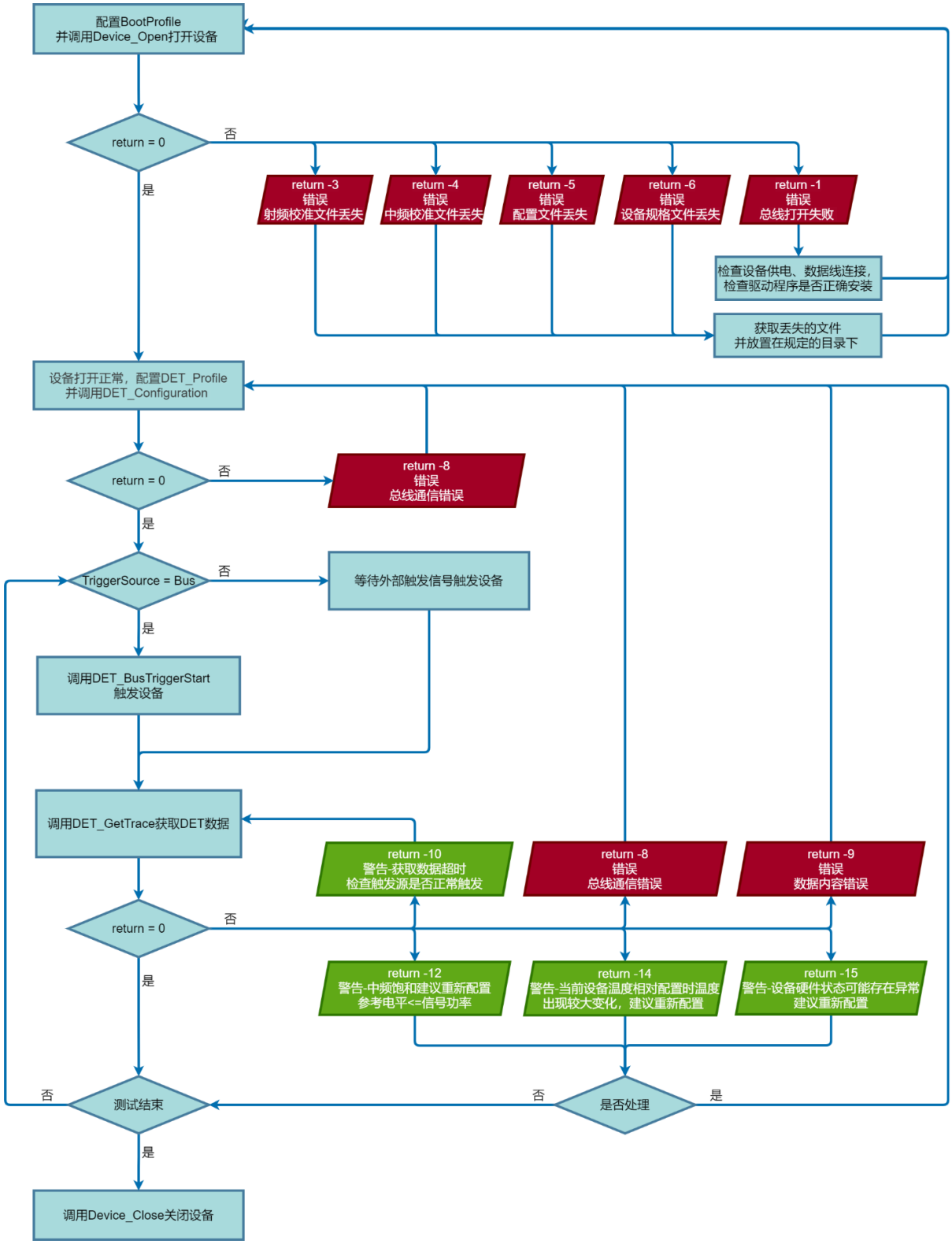


图5　DET 模式调用流程图(触发模式为 Fixed)

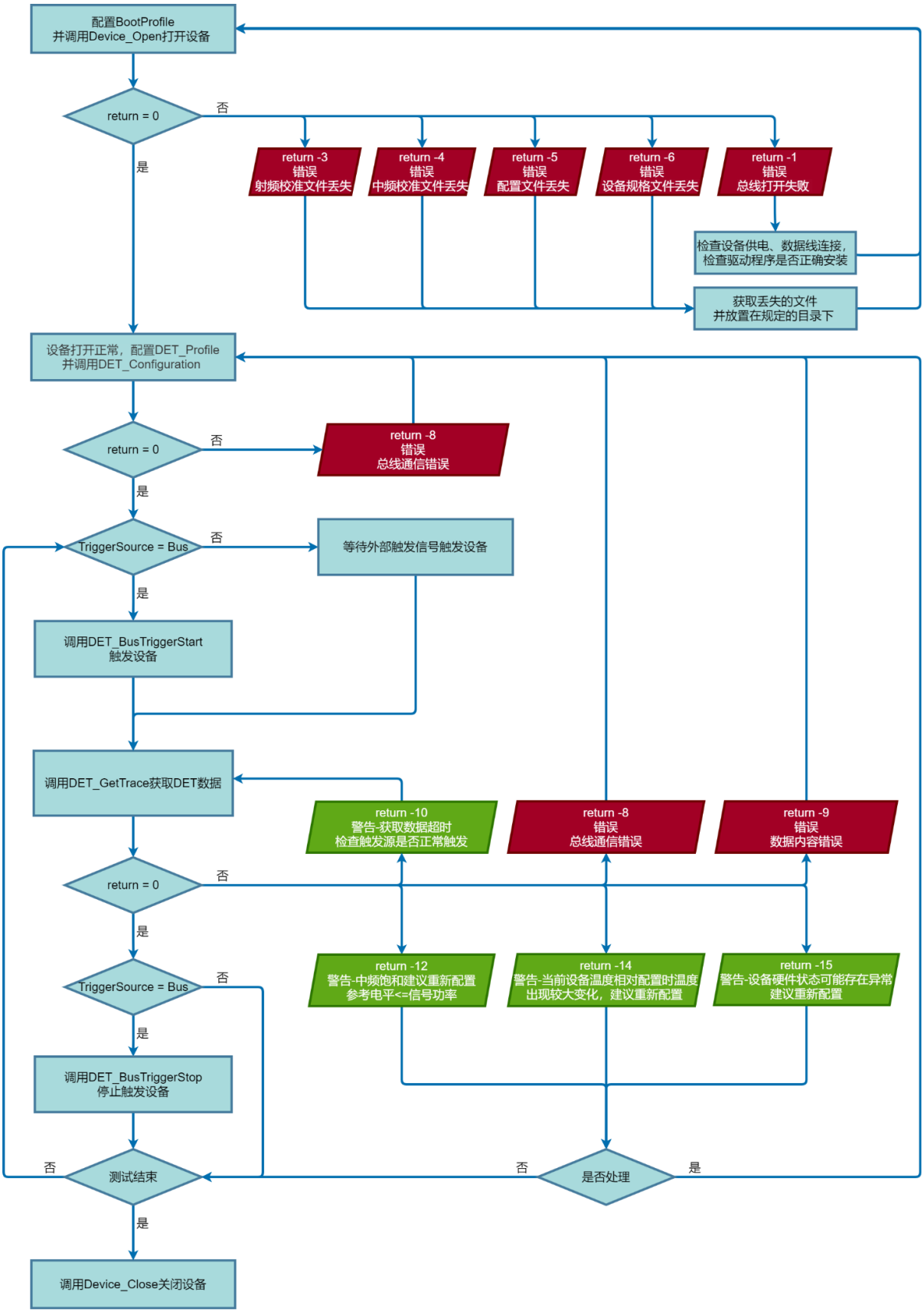


图6　DET 模式调用流程图(触发模式为 Adaptive)

6.4 实时频谱分析（RTA）的 API 调用地图

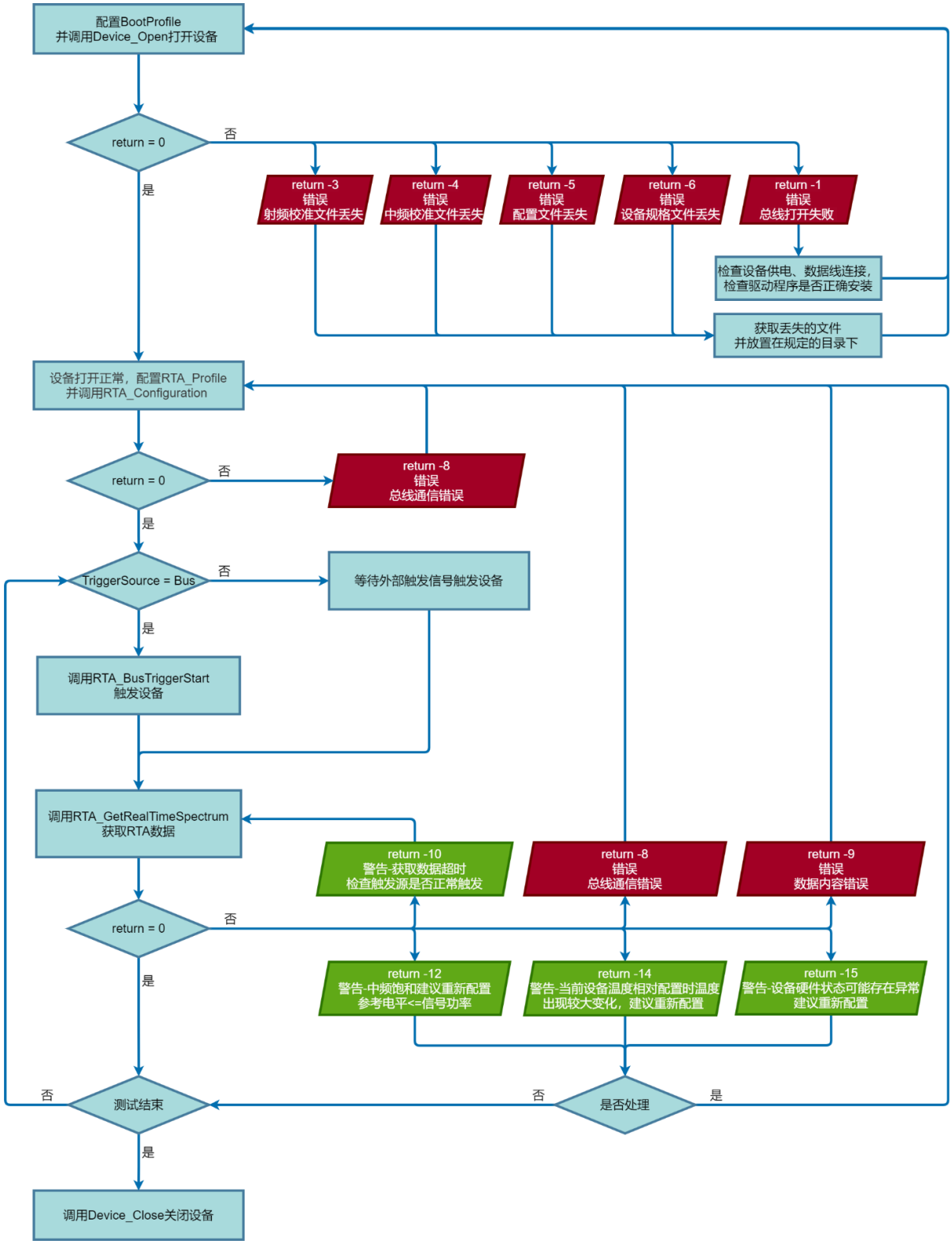


图7　RTA 模式调用流程图(触发模式为 Fixed)

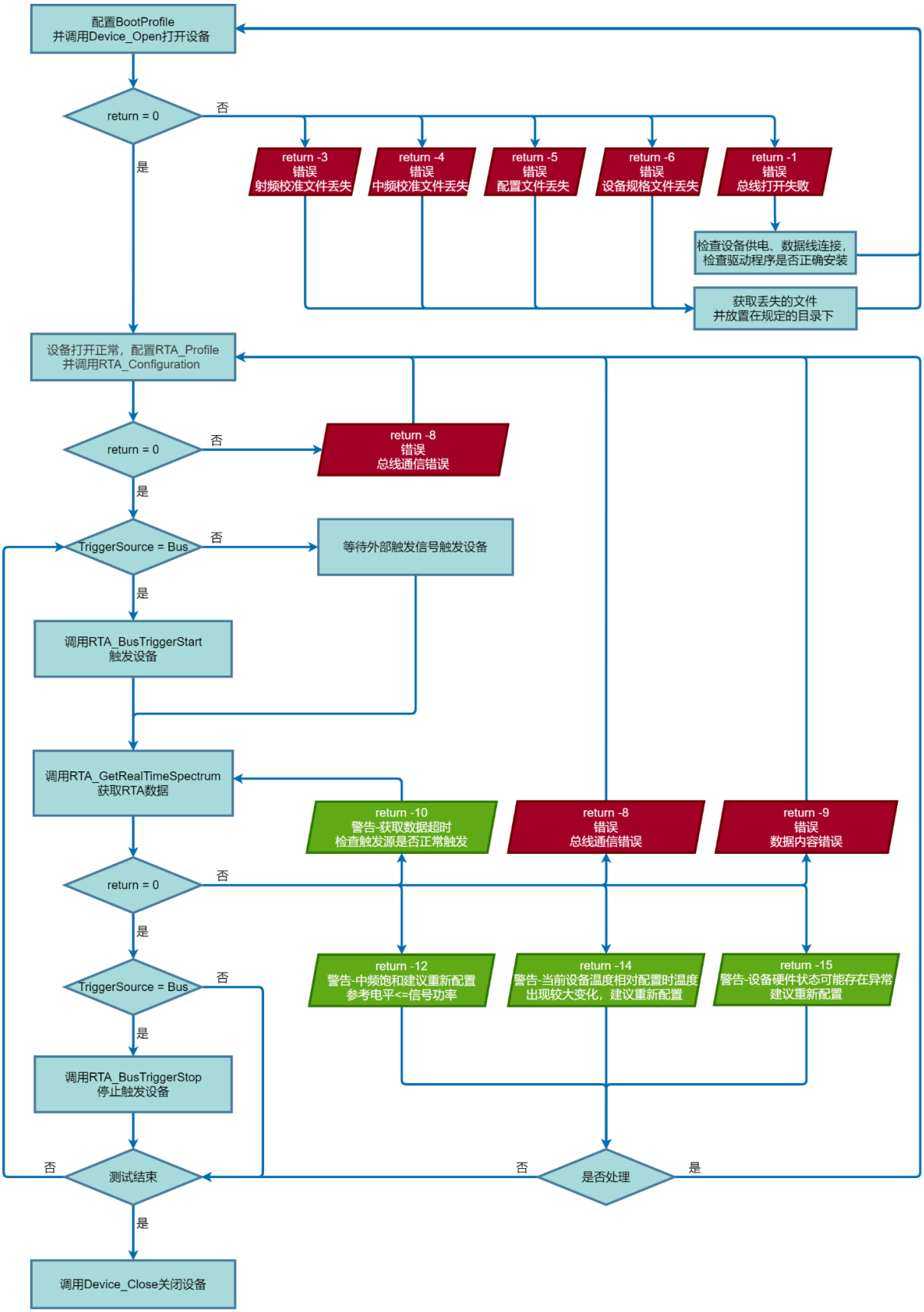


图8　RTA 模式调用流程图(触发模式为 Adaptive)

7 重要变量/重要设置及其概念

此章节罗列了频谱分析仪/接收机产品所涉及到的部分重要参数及其概念，充分理解这些参数与概念对于正确并高效地使用设备具有重要意义。在此汇总以便于预览或遇到问题时查阅。

7.1 系统

表4 系统参数与相关概念信息说明表

序	参数与概念	适用模式	说明
1	设备内存指针 void **Device	Device/SWP/IQS/DET/RTA	此参数为设备运行所需的内存空间引用。调用API时，必须通过此引用来索引此次打开的设备。
2	设备ID DeviceUID	Device	每个设备具有唯一的设备ID标识，请使用该ID用于区分不同的设备个体。

7.2 幅度

表5 幅度参数与相关概念信息说明表

序	参数与概念	适用模式	说明
1	参考电平 RefLevel_dBm	SWP/IQS/DET/RTA	系统默认使用自动增益设置，衰减器（Atten）与前置放大器（Preamplifier）由系统根据参考电平自动配置。在SWP模式下，参考电平表示在所设置的频率范围内，系统保持准确读数所能接受的最大输入功率。例如，设置起始频率100MHz，终止频率1GHz，参考电平为0dBm，则此时只要输入的频率位于100MHz~1GHz之间、信号功率不高于0dBm，系统可确保读数不会因为出现压缩而失准。参考电平亦可以简单理解为系统不饱和所能接受的最大输入功率。对于定频点的模式，如IQS/DET/RTA模式，参考电平适用于所设置的中心频率。系统在处理参考电平时会保留一定裕量，一般为1~6dB，所以在一些频率处，即使输入功率大于参考电平，系统仍然不会报告饱和警告，此为正常现象。根据预期的输入功率设置参考电平，使得参考电平稍高于预期的输入信号最大功率，一般可以获得良好的动态范围。比如预期信号为单音信号，功率为-3dBm，则设置参考电平为0dBm，可获得良好的观察动态。
	衰减 Atten		衰减默认为自动（Atten = -1），此时系统完全由RefLevel_dBm指定通道增益。当需要手动设置通道增益时，请设置Atten为指定值。

2	前置放大器 Preamplifier	SWP/IQS/DET/RTA	对于配置有前置放大器的设备，前置放大器布置于衰减器之前，位于系统最前级，其是否开启对于系统噪声与线性度具有显著影响。开启前置放大器可降低噪声，但会降低系统的最大线性输入功率与最大损毁功率。可指定系统根据参考电平自动开启前置放大器或任何情况下都关闭前置放大器（避免过载损坏）。
3	模拟中频带宽 AnalogIFBWGrade	SWP/IQS/DET/RTA	对于配置有多个模拟中频滤波器的设备，系统提供多个不同特性的中频通道供选择。不同的中频档位具有不同的增益、带宽、群延时等性能，请根据应用需要选择合适的中频档位。
4	中频增益档位 IFGainGrade	SWP/IQS/DET/RTA	系统允许用户设置中频增益，以选择性优化杂散、线性度及噪声水平，档位序号越高中频增益越高，一般每个档位之间相差1dB~3dB。系统增益包括射频增益与中频增益两部分，当总增益保持固定时（参考电平不变），提高中频增益将降低接收机的混频器输入功率，有助于改善杂散但会恶化噪声性能，反之降低中频增益将提高接收机的混频功率，可能会恶化杂散但改善噪声性能。当射频增益处于最大状态时（有效参考电平达到最低值），提高中频增益可以进一步提高整机增加，并可能改善噪声性能。例如，当参考电平为0dBm时（此时远未达到系统射频最高增益），提高中频增益，将优化杂散与线性度但恶化噪声，降低中频增益，将恶化杂散与线性度但改善噪声；当参考电平为-60dBm时（此时系统射频增益达到最大值），提高中频增益可能优化噪声。

7.3 频率

表6　频率参数与概念相关信息说明表

序	参数与概念	适用模式	说明
1	频率指定方式 FreqAssignment	SWP	SWP模式下，允许用户通过此变量以StartStop方式或CenterSpan方式来指定频率扫描范围。
2	起始、终止频率 中心频率与扫宽 StartFreq_Hz StopFreq_Hz CenterFreq_Hz Span_Hz		

3	迹线点数策略 TracePointStrategy	SWP	SWP模式下，使用完全基于FFT分析方法实现频谱分析。当以扫描速度优化的方式执行频率扫描时（TracePointStrategy = SweepSpeedPreferred），由于底层软件的实现限制与迹线检波的存在，系统无法使原生的分析频率点与指定的起始终止频率对应，且返回的数据会在小幅超出所设置的频率范围，用户可根据需要，做一定的截取。 用户可设置期望的迹线点数（TracePoints），由于存在前述的实现限制，实际迹线点数通常无法达到期望的设置值，系统将返回就近的实际可用点数。
4	迹线点数 TracePoints		
5	相邻频点之间频率间隔（未生效） TraceBinSize_Hz		
6	迹线对齐（未生效） TraceAlign		

7.4 分析

表7　分析参数与相关概念信息说明表

序	参数与概念	适用模式	说明
1	杂散抑制 SpurRejection	SWP	支持关闭、标准、增强三种杂散抑制模式，杂散抑制功能可对大部分组合分量杂散进行有效抑制（对系统剩余响应无优化效果）但会降低扫描速度与时变信号的测量能力。对于稳态信号测试（如单音信号），开启杂散抑制功能可以有效提高测量的无杂散动态范围；对于快速时变信号测试（如调制信号），开启杂散抑制功能可能会导致信号概率性丢失或功率值不准确，需要谨慎开启。通过切换关闭与开启状态，并观察频谱情况，有助于判断当前测试场景是否可以开启此功能。
2	功耗平衡 PowerBalance	SWP	SWP模式下，用户可通过设置功耗平衡参数在 扫描速度与设备功耗之间进行取舍，功耗平衡为0时系统将以最高扫描速度允许，非0时的设置范围一般为40~1000，数值越大扫描速度越低同时功耗越低。需要特别注意，在开启杂散抑制的情况下，功耗平衡控制会导致系统对时变信号检测能力的进一步下降，功耗平衡数值越高，检测能力下降越明显。对于高时变信号检测的应用场景，需要谨慎设置。
3	窗型 Window	SWP/RTA	在执行基于FFT的频谱分析时，系统提供多种窗型，不同窗型具备不同的优势，请根据需要选择。例如，FlatTop窗具有良好的幅度准确性，可降低由于栅栏效应引起的幅度精度损失，适合用于高幅度精度要求的测试场合；Blackman-Nuttall窗具有窄的主瓣宽度，频率分辨率高，相同RBW情况下，其分析速度快于FlatTop窗，适合于高频率分辨力及快速扫描的场景。
4	迹线检波、迹线点数 Detector, TracePoints	SWP/RTA	原生的频谱迹线可能具有上万甚至数十万的数据点，检波用于将频谱迹线按照指定的方式进行压缩，将数据规模控制在应用所需的范围内。由于系统在FFT分析与帧拼接上底层软硬件的限制，系统无法设置任意的迹线点数，而是根据用户输入的期望点数自动选取就近可用但不大于期望点数的可用设置进行配置。所以在每次配置时，请务必留意配置函数的返回信息，以获得实际生效的迹线点数。
5	FFT执行策略	SWP	在标准频谱分析模式下，用户可选择由系统根据RBW自动

	FFTExcutionStrategy		选择FPGA运算还CPU运算或者仅由FPGA或CPU计算。FPGA计算可大幅降低CPU处理能力要求，但在中小RBW情况下（RBW ≤ 5kHz）由单次FFT点数限制，扫描速度较慢；CPU计算允许系统使用超过64k点的FFT点数，在中小RBW情况下可以获得比FPGA更高的扫描速度。
6	帧时间倍率 FrameTimeMultiple	SWP	故此，使用严格确定的 在单一频点上的采样时长 作为扫描时间的定义。用户在调用SWP_Configuration函数时，系统会返回当前配置下的 扫描时间。对于需要延长扫描时间的应用，可以通过设置SweepTimeMultiple （表示采样时长相对最小时长的倍数）来延长系统在单个频点上的采样时长。在RBW = VBW时，延长扫描时间后，系统将返回采样窗口中多个FFT帧中总功率最高的那一帧作为结果输出，这个特性有助于用户在分析脉冲等时变信号时，获取准确的信号功率。
7	帧检波器 FrameDetector	SWP/RTA	在同一个本振频点下存在多帧频谱分析结果时，允许以用户指定的规则（帧检波器）将多帧结果转化为需要的输出的形式。
8	扫描时间 SweepTime		本系统中定义扫描时间为完成一次从起始频率至终止频率扫描所需要的总时间。 当设置SweepTime = -1或小于预计的最小扫描时时（TraceInfo.EstimatedMinSweepTime），可指定系统总是以最高的扫描速度进行扫描。由于系统扫描一条迹线所需的时间与多个不确定因素有关（CPU占用率、代码设计等），在最高扫描速度设置下，系统无法直接指定获取一条迹线所需的总时间，只能给出估计值。 当设置较大的扫描时间时（SweepTime > TraceInfo.EstimatedMinSweepTime），系统将对扫描时间进行准确控制。
9	抽取倍数 DecimateFactor	IQS/DET/RTA	IQS/DET/RTA模式下，系统使用抽取倍数来设置分析带宽。 分析带宽 = 抽取倍数为1时的分析带宽 / 抽取倍数。由于底软软硬件限制，抽取倍数无法任意可设，系统会根据期望的抽取倍数选择就近可用的值进行配置并反馈用户。
10	总线超时 BusTimeOut	IQS/DET/RTA	总线超时为获取数据的相关函数设置了一个执行时间上限，系统若在该时间内无法获取有效数据，则会强制返

			回，以免系统无限制地等待。在IQS/DET/RTA模式下，需要设置。

7.5 默认单位

表8　主要变量单位汇总表

变量	单位
频率	Hz
功率	dBm
电压	V
时间	s

8 设备与系统 Device 主要函数

在调用任何设备硬件关联的API函数前都需要首先调用Device_Open函数对设备进行打开，并在完成业务程序之后，调用Device_Close函数对设备进行关闭，以释放内存空间。

8.1 Device_Open

int Device_Open(void Device, int DeviceNum, const BootProfile_TypeDef* BootProfile BootInfo_TypeDef* BootInfo)**

功能描述

未打开的设备需要被打开才能进行后续API的调用。调用本函数以打开设备，函数将返回一个句柄，该句柄用于在后续API调用中指向目标设备。当有多台设备时，通过指定不同的设备号（DeviceNum）来分别打开对应的设备，并获取其设备句柄。在后续API调用中，使用设备句柄来指定哪台设备被操作。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。在调用其它API时，通过此句柄来索引需要调用的设备。
int DeviceNum	指定设备号，当主机连接有多台设备时，可通过此设备号来选择相应的设备，设备号从0开始累加。
BootProfile_TypeDef *BootProfile	设置启动配置。
BootInfo_TypeDef *BootInfo	启动信息反馈。
BootProfile_TypeDef 详细定义	
PhysicalInterface_TypeDef PhysicalInterface	指定以何种物理端口来尝试设备的开启，必须正确设置才能正常开启设备，否则将无法打开设备。一台设备可能配备多种接口。 USB：使用USB接口进行数据传输。 QSFP：使用40Gbps QSFP+ 进行数据传输； ETH：使用100M/1000M 以太网进行数据传输； HLVDS：使用LVDS总线进行数据传输； VIRTUAL：使用虚拟总线进行仿真、调试。
DevicePowerSupply_TypeDef DevicePowerSupply	指定设备的供电方式，必须正确设置才能正常开启设备，否则将无法打开设备。 USBPortAndPowerPort：使用USB数据端口及独立电源端口双供电； USBPortOnly：仅使用USB数据端口供电； Others：当使用非USB总线时，比如ETH，使用此选项。
IPVersion_TypeDef	以太网IP版本：

ETH_IPVersion	IPv4：使用IPv4地址； IPv6：使用IPv6地址。
uint8_t ETH_IPAddress[16]	当物理接口为ETH时，用于指定IP地址，例如目标IP地址为192.168.1.100，ETH_IPAddress[0] = 192；ETH_IPAddress[1] = 168；ETH_IPAddress[2] = 1；ETH_IPAddress[3] = 100； 当网络协议为IPv4时，只需要填前4个数即可，数组其余部分不需要填。 IPv6暂未支持。
uint16_t ETH_RemotePort	当物理接口为ETH时，指定侦听端口。
int32_t ETH_ErrorCode	当物理接口为ETH时，返回连接过程中的错误代码。
int32_t ETH_ReadTimeOut	当物理接口为ETH时，设定通信的超时时间。当设备配置与数据函数在此时间内未响应时，函数将强制返回并返回错误代码。

BootInfo_TypeDef 详细定义

DeviceInfo_TypeDef DeviceInfo	设备信息
uint32_t BusSpeed	总线带宽信息
uint32_t BusVersion	总线固件版本。
uint32_t APIVersion	当前API版本。采用主版本，子版本，修订版本表示。 bit[31..16]表示主版本，bit[15..8]表示子版本，bit[7..0]表示修订。
int ErrorCodes[7]	启动过程中的错误代码清单。
int Errors	启动过程中的错误总数。
int WarningCodes[7]	启动过程中的警告代码清单。
int Warning	启动过程中的警告总数。

DeviceInfo_TypeDef 详细定义

uint64_t DeviceUID	设备独有的ID号。
uint16_t Model	设备型号。
uint16_t HardwareVersion	设备硬件版本。
uint16_t MFWVersion	设备主控固件版本。
uint16_t FFWVersion	设备FPGA固件版本。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	必须在其它函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的Device_Open调用，必须在整个模块使用需求结束之后，调用Device_Close函数，以释放内存。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
```

8.2 Device_Close

Device_Close(void Device)**

功能描述

此函数关闭已打开频谱仪设备，在结束调用设备时需要调用此函数已关闭USB设备及释放设备所开辟的内存空间。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后USB设备连接关闭，内存空间释放。如需要重新使用设备，则需要再次通过调用Device_Open，来建立USB连接及打开设备。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
Status = Device_Close(&Device);
```

8.3 Device_QueryDeviceState

int Device_QueryDeviceState(void Device, DeviceState_TypeDef* DeviceState)**

功能描述

此函数获取当前频谱仪的设备状态信息，包括设备温度、硬件工作状态、地理时间信息（需要选件支持）等，非实时方式，不打断数据获取，但信息只在获取数据包后更新。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
DeviceState_TypeDef	当前设备状态相关信息的结构体指针。调用此函数后，该指针指向的相

***DeviceState**	关信息会被更新成最新的值。
DeviceState_TypeDef 详细定义	
int16_t Temperature	设备的温度。
uint16_t RFState	设备射频状态。
uint16_t BBState	设备基带（BB）状态。
double AbsoluteTimeStamp	绝对时间戳。
float Latitude	纬度坐标,北纬为正数，南纬为负数，以此区分南北纬。
float Longitude	经度坐标,东经为正数，西经为负数，以此区分东西经。
uint16_t GainPattern	增益控制字。
int64_t RFCFreq	射频中心频率。
uint32_t ConvertPattern	频率变换方式。
uint32_t NCOFTW	NCO频率字。
uint32_t SampleRate	等效采样率，等效采样率 = ADC原始采样率/抽取倍数。
uint16_t CPU_BCFlag	非用户信息。CPU-FFT做拼帧时，所需的BC标志位。
uint16_t IFOverflow	指示设备是否中频过载。 0：未过载；1：过载，测量结果可能不准确
uint16_t DecimateFactor	抽取倍数
uint16_t OptionState	指示选件状态
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceState_TypeDef *DeviceState;
Status = Device_QueryDeviceInfo(&Device, &DeviceState);
```

8.4 Device_SetIPAddr

int Device_SetIPAddr(void Device, const IPVersion_TypeDef ETH_IPVersion, const uint8_t ETH_IPAddress[], const uint8_t SubnetMask[])**

功能描述

在ETH接口的设备中（如NX系列产品），调用此函数可以修改IP地址。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
IPVersion_TypeDef	设置IP地址版本,为枚举类型：
ETH_IPVersion	IPv4:使用IPv4地址；
	IPv6:使用IPv6地址。
const uint8_t ETH_IPAddress[]	以字符串设置IP地址。
const uint8_t SubnetMask[]	以字符串设置网关地址。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
const IPVersion_TypeDef  ETH_IPVersion = 0;
const uint8_t ETH_IPAddress[4] = {192,168,1,100};
const uint8_t SubnetMask[4] ={192,168,1,1};
Status = Device_ChangeIPAddr(&Device, ETH_IPVersion, ETH_IPAddress, SubnetMask);
```

9 设备与系统 Device 其他函数

9.1 Device_QueryDeviceInfo

int Device_QueryDeviceInfo(void Device, DeviceInfo_TypeDef* DeviceInfo)**

功能描述

此函数获取设备信息，包括设备序列号及软硬件版本等相关信息，非实时方式，不打断数据获取，但信息只在获取数据包后更新。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
DeviceInfo_TypeDef *DeviceInfo	返回当前设备序列号及软硬件版本等相关信息的结构体指针。调用此函数后，该指针指向的相关信息会被更新成最新的值。
DeviceInfo_TypeDef 详细定义	
uint64_t DeviceUID	设备独有的ID号。
uint16_t Model	设备型号。
uint16_t HardwareVersion	设备硬件版本。
uint16_t MFWVersion	设备主控固件版本。
uint16_t FFWVersion	设备FPGA固件版本。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo(&Device, &DeviceInfo);
```

9.2 Device_QueryDeviceInfo_Realtime

int Device_QueryDeviceInfo_Realtime(void Device, DeviceInfo_TypeDef* DeviceInfo)**

功能描述

此函数获取设备信息，包括设备序列号及软硬件版本号等相关信息，实时获取，短时间内会占用数据通道，但可以保证总是获得即时的设备信息。

兼容性	0.52.0及之后版本支持
参数说明	同函数Device_QueryDeviceInfo
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceInfo);
```

9.3 Device_QueryDeviceState_Realtime

int Device_QueryDeviceState_Realtime(void Device, DeviceState_TypeDef* DeviceState)**

功能描述

此函数获取设备状态，包括设备温度、硬件工作状态、地理时间信息（需要选件支持）等，实时获取，短时间内会占用数据通道。

兼容性	0.52.0及之后版本支持
参数说明	同函数Device_QueryDeviceState
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
```

```
DeviceState_TypeDef *DeviceState;
Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceState);
```

9.4 Device_UpdateFirmware

int Device_UpdateFirmware (int DeviceNum, const char *path)

功能描述

此函数更新MCU设备固件。

兼容性	0.52.0及之后版本支持
参数说明	
int DeviceNum	指定设备号，当主机连接有多台设备时，可通过此设备号来选择相应的设备，设备号从0开始累加。
const char *path	指定待写入固件所在的文件地址。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	可直接运行此函数对设备固件进行升级。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
char *path = "D:\\ MCUFirmware.bin";
Status = Device_MCUFirmwareUpdate(&Device, path);
```

9.5 Device_SetSysPowerState

int Device_SetSysPowerState(void **Device, SysPowerMode_TypeDef SysPowerMode)

功能描述

此函数设置设备的电源状态，包括上电运行、射频部分下电、射频部分待机等。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
SysPowerMode_TypeDef SysPowerMode	设备功耗控制： PowerOn: 系统所有工作区均上电； RFPowerOFF: 射频处于下电状态，不可快速唤醒； RFStandby: 射频处于待机状态，可快速唤醒。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SysPowerMode_TypeDef SysPowerMode = PowerON;
Status = Device_SetSysPowerMode(&Device, SysPowerMode);
```

9.6 Device_SetFanState

int Device_SetFanState(void Device, FanModeCtrl_TypeDef FanModeCtrl, float Temperature)**

功能描述

此函数控制设备风扇工作模式。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
FanModeCtrl_TypeDef FanModeCtrl	风扇模式： FAN_FORCE_ON:强制常开； FAN_FORCE_OFF:强制长关； FAN_AUTO:自动模式。
float Temperature	门限温度（摄氏度），当FanModeCtrl = FAN_AUTO时，若设备温度高于此温度，系统启动风扇，并在低于此温度-5℃时，关闭风扇
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
FanModeCtrl_TypeDef FanModeCtrl = 0;
float Temperature = 0;
Status = Device_FanModeCtrl(&Device, FanModeCtrl, Temperature);
```

9.7 Device_CalibrateRefClock

int Device_CalibrateRefClock(void Device, ClkCalibrationSource_TypeDef ClkCalibrationSource, const double TriggerPeriod_s, const uint64_t TriggerCount, const bool RewriteRFCal, double* RefCLKFreq_Hz)**

功能描述

此函数通过外触发信号或系统内GNSS1PSS校准参考时钟频率。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
ClkCalibrationSource_TypeDef ClkCalibrationSource	选择校准所使用的定时信号源： CalibrateByExternal: 通过Ext触发进行Clock校准； CalibrateByGNSS1PPS = 0x01: 通过GNSS-1PPS进行Clock校准
const double TriggerPeriod_s	输入已知的校准信号的准确周期，该值精度将直接影响校准效果。
const uint64_t TriggerCount	指定用于校准的触发次数，对于使用GNSS1PPS进行校准的场景，触发次越数多，通常对于抖动误差的抑制就越好，校准效果越好，但校准时间也越长。使用GNSS1PSS时，建议触发次数大于30，即校准超过30秒。
const bool RewriteRFCal	0：校准结果不写入校准文件，设备下电后校准结果失效。 1：校准结果写入校准文件，设备上下电后仍然保持校准状态。
double* RefCLKFreq_Hz	反馈本次校准得到的新的参考时钟频率，即校准结果。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0;void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
const double ExtTriggerPeriod_s = 1;
uint64_t CalibrationTimes = 1 * 60;
bool RewriteRFCal = false;
double RefCLKFreq_Hz = 0;
Status = Device_CalibrateRefClock(&Device, ExtTriggerPeriod_s, CalibrationTimes, RewriteRFCal,&
RefCLKFreq_Hz);
```

9.8 Device_Reboot

int Device_Reboot(void Device)**

功能描述

在ETH接口的设备中（如NX系列产品），调用此函数重启设备。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO; BootInfo_TypeDef *BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
Status = Device_Reboot(&Device);
```

9.9 Device_RestartNetwork

int Device_RestartNetwork(void Device)**

功能描述

在ETH接口的设备中（如NX系列产品），调用此函数重启网络，使得修改的IP地址生效。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在Device_Open后调用此函数，若调用Device_ChangeIPAddr函数修改地址，需在后调用此函数才会生效。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
```

```
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
const IPVersion_TypeDef  ETH_IPVersion = 0;
const uint8_t ETH_IPAddress[4] = {192,168,1,12};
const uint8_t SubnetMask[4] ={192,168,1,1};
Status = Device_ChangeIPAddr(&Device, ETH_IPVersion, ETH_IPAddress, SubnetMask);
Status = Device_RestartNetwork(&Device);
```

10 标准频谱分析 SWP 主要函数

10.1 SWP_ProfileDeInit

int SWP_ProfileDeInit(void Device, SWP_Profile_TypeDef* UserProfile_O)**

功能描述

此函数初始化配置SWP模式配置参数集合（SWP_Profile_TypeDef）。SWP_Profile_TypeDef定义了设备在SWP模式下的频率、参考电平、分辨率带宽等所有参数。

兼容性	0.52.0 及之后版本支持
参数说明	
void **Device	设备句柄。
SWP_Profile_TypeDef **UserProfile_O	输出默认的配置参数集合。
SWP_Profile_TypeDef 详细定义	
double StartFreq_Hz	起始频率，单位Hz。
double StopFreq_Hz	终止频率，单位Hz。
double CenterFreq_Hz	中心频率，单位Hz。
double Span_Hz	频率扫宽，单位Hz。
double RefLevel_dBm	参考电平，单位dBm。
double RBW_Hz	分辨率带宽，单位Hz。
double VBW_Hz	视频带宽，单位Hz。
double SweepTime	当前版本未生效。扫描时间，单位秒。指定完成一次完整迹线扫描所需要的时间，当输入值小于设备最小扫描时间时，系统将仍然以最小时间进行扫描。
double TraceBinSize_Hz	迹线相邻频点之间的频率间隔，单位Hz。仅在TracePointsStrategy = BinSizeAssigned 时生效。
SWP_FreqAssignment_TypeDef **FreqAssignment**	设置频率的指定方式，选择以StartStop或CenterSpan设定频率。 StartStop：以起始频率、终止频率指定扫描范围。 CenterSpan：以中心频率、扫宽指定扫描范围。
Window_TypeDef **Window**	指定FFT分析所使用的窗函数： FlatTop：具有良好的幅度准确度。 Blackman_Nuttall：具有良好的频率分辨力。
uint32_t TracePoints	设置期望的迹线点数，系统会根据设置的扫描范围和RBW进行自动调整，并返回实际可用且最接近的迹线点数。

TracePointsStrategy_TypeDef **TracePointsStrategy**	设置迹线点数的设置策略： SweepSpeedPreferred：优先保证扫描速度最快，然后尽量靠近设置的目标迹线点数； PointsAccuracyPreferred：优先保证实际迹线点数接近设置的目标迹线点数； BinSizeAssigned：优先保证按照设定的频率间隔来生成迹线。
TraceAlign_TypeDef **TraceAlign**	设置迹线对齐的方式： NativeAlign：自然对齐； AlignToStart：对齐至起始频率； AlignToCenter：对齐至中心频率。
FFTExcutionStrategy_TypeDef **FFTExcutionStrategy**	设置FFT执行策略： Auto：根据设置自动选择使用CPU还是FPGA进行FFT计算； Auto_CPUPreferred：根据设置自动选择使用CPU还是FPGA进行FFT计算，CPU优先； Auto_FPGAPreferred：根据设置自动选择使用CPU还是FPGA进行FFT计算，FPGA优先； CPUOnly_LowResOcc：强制使用CPU计算，低资源占用，最大FFT点数256K； CPUOnly_MediumResOcc：强制使用CPU计算，中资源占用，最大FFT点数1M； CPUOnly_HighResOcc：强制使用CPU计算，高资源占用，最大FFT点数4M； FPGAOnly：强制使用FPGA计算。
DetMode_TypeDef **Detector**	设置迹线检波器类型： PosPeak：正峰值；NegPeak：负峰值；Sample：取样；Normal：正态；RMS：均方根值; NoneDet：无迹线检波
RxPort_TypeDef **RxPort**	设置信号接收端口：： ExternalPort：接收机接收来自外部输入端口输入的数据； InternalPort：接收机接收来自内部的辅助信号源的射频信号； ANT_Port、TR_Port、SWR_Port、INT_Port：仅用于TRX产品
SpurRejection_TypeDef **SpurRejection**	设置杂散抑制程度： Bypass：没有杂散抑制； Standard：标准杂散抑制； Enhanced：增强杂散抑制； 杂散抑制等级越高，扫描速率越慢；

ReferenceClockSource_TypeDef **ReferenceClockSource**	设置参考时钟源： ReferenceClockSource_Internal：内部参考时钟（默认10MHz）； ReferenceClockSource_External：外部参考时钟（默认10MHz），当外部参考无法锁定时将自动切换为内部参考； ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：强制使用外部参考时钟，即使无法锁定也不会切换至内部参考；
double ReferenceClockFrequency	设置参考时钟频率 Hz，可用手动对系统时钟准确度进行修正。
SystemClockSource_TypeDef **SystemClockSource**	设置系统时钟源，默认使用内部系统时钟，请在厂商指导下使用： SystemClockSource_Internal: 内部系统时钟源； SystemClockSource_External: 外部系统时钟源；
Double **ExternalSystemClockFrequency**	设置外部系统时钟频率，Hz。
float FrameTimeMultiple	帧分析时间倍率，设备在单一频点上的分析时间 = 默认分析时间（系统自行设定） * 帧时间倍率。提高帧时间倍率将增加设备的最小扫描时间，但不严格线性。
FrameDetector_TypeDef **FrameDetector**	帧检波检波器，对同一个频点下的多次FFT分析之间进行运算处理： FrameDetector_Bypass：不进行帧间检波； FrameDetector_MaxHold：取最大值保持后输出； FrameDetector_Average：取线性平均后输出； FrameDetector_MinHold：取最小值保持后输出； FrameDetector_MaxPower：取总功率最大的帧输出； FrameDetector_RawFrames：对所有帧进行逐帧输出。
SWP_TriggerSource_TypeDef **TriggerSource**	设置RF接收机的扫频触发源： InternalFreeRun：自动连续扫描； ExternalPerHop：外部触发触发单帧扫描； ExternalPerSweep：外部触发触发一次完整迹线扫描； ExternalPerProfile：暂未开放 。
TriggerEdge_TypeDef **TriggerEdge**	设置触发边沿： RisingEdge：由上升沿触发； FallingEdge：由下降沿触发； DoubleEdge：由双边沿触发。
TriggerOutMode_TypeDef**Trigger** **OutMode**	设置触发输出的模式： None：无触发输出；

	PerHop：随每帧完成时输出； PerSweep：随每次扫描完成时输出； PerProfile：随每次配置切换输出。
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	设置触发输出的脉冲极性： Positive：正脉冲； Negative：负脉冲。
uint32_t PowerBalance	设置SWP模式下的动态功耗控制。典型范围为0~5000，增大该值可降低功耗但会降低扫描速度。
GainStrategy_TypeDef **GainStrategy**	设置增益策略： LowNoisePreferred：侧重低噪声； HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef **Preamplifier**	设置前置放大器动作： AutoOn：自动使能前置放大器； ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。
uint8_t EnableDebugMode	设置使能调试模式。
uint8_t CalibrationSettings	校准设置，仅调试或高级应用使用，默认值为0。
int8_t Atten	设置衰减dB,设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
SWP_TraceType_TypeDef **TraceType**	设置频率迹线类型： ClearWrite：清除写入； MaxHold：最大值保持； MinHold：最小值保持； ClearWriteWithIQ：清除写入并附带对应的IQ时域数据。
SWP_RBWMode_TypeDef **RBW_Mode**	设置RBW的配置模式： RBW_Manual：手动设置RBW； RBW_Auto：自动设置RBW。
VBWMode_TypeDef **VBW_Mode**	设置VBW配置模式： VBW_Manual:手动输入VBW； VBW_EqualToRBW:强制VBW=RBW； VBW_TenPercentRBW:强制VBW=0.1*RBW； VBW_OnePercentRBW:强制VBW=0.01*RBW。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在Device_Open后调用。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO; BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
```

10.2 SWP_Configuration

int SWP_Configuration(void Device, const SWP_Profile_TypeDef* ProfileIn, SWP_Profile_TypeDef* ProfileOut, SWP_TraceInfo_TypeDef* TraceInfo)**

功能描述

此函数将频谱仪设备配置成标准的频谱扫扫描模式（SWPMode）并配置SWP模式的相关参数。SWP模式下的频率、参考电平、分辨率带宽等参数统一封装在SWP_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
SWP_Profile_TypeDef *SWP_ProfileIn	配置集输入。
SWP_Profile_TypeDef *SWP_ProfileOut	配置集输出。
SWP_TraceInfo_TypeDef *TraceInfo	返回频谱迹线的相关信息。

SWP_TraceInfo_TypeDef 详细定义

int FullsweepTracePoints	一次完整频谱扫描包含全部频谱数据点数。
int PartialsweepTracePoints	一个扫描片段（帧）所包含的频谱数据点数，多个扫描片段（TotalHops）构成一次完整的扫描。
int TotalHops	为一次完整扫描包含的扫描片段/帧个数。
uint32_t UserStartIndex	迹线数组中与用户指定StartFreq_Hz对应的数组索引，即HopIndex = 0时，Freq[UserStartIndex]是与SWPProfile.StartFreq_Hz最为近的频率点。
uint32_t UserStopIndex	迹线数组中与用户指定SttopFreq_Hz对应的数组索引，即HopIndex = TotalHops - 1时，Freq[UserStopIndex]是与SWPProfile.StopFreq_Hz最为近的频率点。
double TraceBinBW_Hz	实际迹线两点之间的频率间隔。

double StartFreq_Hz	实际迹线第一个频点的频率。
double AnalysisBW_Hz	每帧的分析带宽。
int TraceDetectRatio	实际的迹线检波比，可指示当前配置下最大可用迹线点数。
int DecimateFactor	当前配置下的抽取倍数。
double FrameTime	实际的帧扫描时间：用于进行单帧FFT分析的信号持续时间（单位为秒）。
double EstimateMinSweepTime	预计的最小扫描时间，即当前配置下所能设定的最小扫描时间（单位为秒，结果主要受Span、RBW、VBW、帧扫描时间等因素影响）。
DataFormat_TypeDef DataFormat	时域数据的数据格式，仅在**TraceType =** ClearWriteWithIQ 时生效。
Uint64_t SamplePoints	时域数据采样长度，仅在**TraceType =** ClearWriteWithIQ 时生效。
uint32_t GainParameter	增益参数信息，包括Space(31~24Bit)、PreAmplifierState(23~16Bit)、StartRFBand(15~8Bit)、StopRFBand(7~0Bit)。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在SWP_ProfileDeInit之后进行调用。

示例

```
int Status = -1;
intDeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = Device_Close(&Device);
```

10.3 SWP_GetPartialSweep

int SWP_GetPartialSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], int* HopIndex, int* FrameIndex, MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

此函数可获取SWP模式下在每个跳频点上获取的频谱结果，同时返回跳频频点序号、帧序号和测量数据的

辅助信息，以便用户自己将多次的扫描的结果拼接成整个频谱曲线，并返回函数状态。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
double Freq_Hz[]	返回的频率数组。数组大小等于通过TraceInfo.PartialSweepTracePoints。
float PowerSpec_dBm[]	返回的幅度数组。数组大小等于通过TraceInfo.PartialSweepTracePoints。
int *HopIndex	返回数据的跳频频点序号。
int* FrameIndex	返回数据的帧序号。
MeasAuxInfo_TypeDef *MeasAuxInfo	返回数据的辅助信息。

MeasAuxInfo_TypeDef 结构体定义

uint32_t MaxIndex	功率最大值在数据包中的索引。
float MaxPower_dBm	数据包中的功率最大值。
int16_t Temperature	设备温度，摄氏度 = 0.01 * Temperature。
uint16_t RFState	射频状态。
uint16_t BBState	基带状态。
uint16_t GainPattern	增益控制字。
uint32_t ConvertPattern	频率变换方式。
double SysTimeStamp	系统时间戳，单位s，设备内定时器提供。
double AbsoluteTimeStamp	绝对时间戳。系统内GNSS提供。
float Latitude	纬度坐标,北纬为正数，南纬为负数，以此区分南北纬。
float Longitude	经度坐标,东经为正数，西经为负数，以此区分东西经。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在SWP_Configuration之后进行调用。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
```

```
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency, PowerSpec_dBm, &HopIndex,
&FrameIndex,MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

10.4 SWP_GetFullSweep

int SWP_GetFullSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

此函数可获取SWP模式下一整条频谱结果和获取测量数据的辅助信息，并同时返回函数状态。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
double Freq_Hz[]	返回的频率数组。数组大小等于通过TraceInfo.FullSweepTracePoints。
float PowerSpec_dBm[]	返回的幅度数组。数组大小等于通过TraceInfo.FullSweepTracePoints。
MeasAuxInfo_TypeDef *MeasAuxInfo	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在SWP_Configuration之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
```

```
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11 标准频谱分析 SWP 其他函数

11.1 SWP_GetPartialSweep_PM1

int SWP_GetPartialSweep_PM1(void Device, SWPTrace_TypeDef* PartialTrace)**

功能描述

SWP_GetPartialSweep的多态形式，在原函数基础上调整了返回数据的形式，加强了数据的封装性。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
SWPTrace_TypeDef *PartialTrace	返回包含配置、返回信息和暂存数据的顶层结构体。
SWPTrace_TypeDef 结构体定义	
double* Freq_Hz	暂存在Device指针中的Frequency数据的地址。
float* PowerSpec_dBm	暂存在Device指针中的PowerSpec_dBm数据的地址。
int HopIndex	数据的跳频频点索引，用于拼接频谱。
int FrameIndex	数据的帧索引，用于准正峰值检波等应用。
void* AlternIQStream	时域数据（交织IQ形式）的地址。
float ScaletoV	时域数据至电压绝对值（V）的系数。
MeasAuxInfo_TypeDef MeasAuxInfo	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。
SWP_Profile_TypeDef SWP_Profile	数据的配置信息。
SWP_TraceInfo_TypeDef SWP_TraceInfo	数据的迹线信息。
DeviceInfo_TypeDef DeviceInfo	数据的设备信息。
DeviceState_TypeDef DeviceState	数据对应的设备状态（定义看htra_api或前文Device_QueryDeviceState函数中）。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在SWP_Configuration之后进行调用。

示例

```
int Status = -1; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
```

```
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
SWPTrace_TypeDef PartialTrace;
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
PartialTrace.Freq_Hz = Frequency;
PartialTrace.PowerSpec_dBm = PowerSpec_dBm;
Status = SWP_GetPartialSweep_PM1(&Device,&PartialTrace);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11.2 SWP_GetPartialUpdatedFullSweep

int SWP_GetPartialUpdatedFullSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], int* HopIndex, int* FrameIndex, MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

此函数可获取长度为完整扫描范围，而片段数据随着每次调用滚动刷新的频谱迹线。适合用于频谱的图形界面显示等应用。

兼容性	0.52.0 及之后版本支持
参数说明	
void **Device	设备句柄。
double Freq_Hz[]	返回的频率数组。数组大小等于通过TraceInfo.FullSweepTracePoints。
float PowerSpec_dBm[]	返回的幅度数组。数组大小等于通过TraceInfo.FullSweepTracePoints。
int* HopIndex	返回数据的跳频频点序号。
int* FrameIndex	返回数据的帧序号。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在SWP_Configuration之后进行调用。
示例	

```
int Status = -1;
int DeviceNum = 0;
```

```
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float *PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialUpdatedFullSweep(&Device, Frequency, PowerSpec_dBm,
&HopIndex,&FrameIndex,&MeasAuxInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

11.3 SWP_MaxHoldReset

void SWP_MaxHoldReset(void** Device)

功能描述

当迹线模式TraceType为 MaxHold或MinHold时，重置保持的迹线数据。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	无返回值。
调用约束	仅在TraceType为 MaxHold或MinHold时生效

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.PartialsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.PartialsweepTracePoints];
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef *MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency, PowerSpec_dBm, &HopIndex,
&FrameIndex,MeasAuxInfo);
SWP_MaxHoldReset(&Device);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

12 IQ 数据记录 IQS 主要函数

IQS模式为 IQ Stream Mode，用户可在此模式下对时域数据流进行连续记录。

12.1 IQS_ProfileDeInit

int IQS_ProfileDeInit(void Device, IQS_Profile_TypeDef* UserProfile_O)**

功能描述

此函数初始化配置IQS模式的相关参数。IQS模式下的中心频率、参考电平、抽取倍数等参数统一封装在IQS_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
IQS_Profile_TypeDef *UserProfile_O	IQS配置结构体指针，为输入/输出变量。
IQS_Profile_TypeDef 详细定义	
double CenterFreq_Hz	设置中心频率。
double RefLevel_dBm	设置参考电平。
uint32_t DecimateFactor	设置抽取倍数，有效分析带宽 = 原始分析带宽/抽取倍数。
RxPort_TypeDef RxPort	设置信号接收端口： ExternalPort：接收机接收来自外部输入端口输入的数据； InternalPort：接收机接收来自内部的辅助信号源的射频信号。
uint32_t BusTimeout_ms	设置数据传输超时时间（ms），如果设备在调用数据获取函数后于此时间内未能成功获取数据则函数返回错误。
IQS_TriggerSource_TypeDef TriggerSource	设置触发源： FreeRun：配置后自动发起单次数据采集； External：由外部触发信号触发一次数据采集； Bus：由函数指令触发一次数据采集； Level：对信号进行连续监测，并在信号功率高于设定的阈值（TriggerLevel_dBm）后进行采集； Timer：由定时器触发数据采集； TxSweep：由信号源扫描触发数据采集； MultiDevSyncByExt：由外部触发同步多设备并触发数据采集； MultiDevSyncByGNSS1PPS：由系统内GNSS提供的1PPS进行多设备同步并触发数据采集；

	SpectrumMask：由频谱模板触发数据采集，仅RTA模式下可用； GNSS1PPS：由系统内GNSS提供的1PPS触发数据采集。
TriggerEdge_TypeDef **TriggerEdge**	设置触发边沿： RisingEdge：上升沿触发； FallingEdge：下降沿触发； DoubleEdge：双边沿触发。
TriggerMode_TypeDef **TriggerMode**	设置触发模式： FixedPoints：触发后，采集指定点数（TriggerLength）的数据； Adaptive：触发后，持续采集数据，直到收到终止信号（外触发终止边沿或总线触发终止指令）。
uint64_t TriggerLength	设置触发长度，即每次触发所采集的数据点数，当TriggerMode = FixedPoints时生效。
TriggerOutMode_TypeDef **TriggerOutMode**	设置触发输出类型： None：无触发输出；
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	设置触发输出脉冲极性： Positive：正脉冲； Negative：负脉冲。
double TriggerLevel_dBm	设置电平触发门限。当TriggerSource = Level时生效。
double TriggerLevel_SafeTime	设置电平触发防抖安全时间，单位为s。若触发信号无法在该时间内保持有效电平，触发将不被执行。当TriggerSource = Level时生效。
double TriggerDelay	设置触发延迟，单位为s。触发后，触发动作将在此延时之后执行。
double PreTriggerTime	设置预触发时间，单位为s。触发后，触发动作前该预触发时长内的数据也将被包含在触发数据中。
TriggerTimerSync_TypeDef **TriggerTimerSync**	设置触发定时器的同步： NoneSync：无同步； SyncToExt_RisingEdge：可被外触发上升沿连续同步； SyncToExt_FallingEdge：可被外触发下降沿连续同步； SyncToExt_SingleRisingEdge：可被外触发上升沿单次同步，调用IQS_SyncTimer_Single函数后开始等待外同步； SyncToExt_SingleFallingEdge：可被外触发下降沿单次同步，调用IQS_SyncTimer_Single函数后开始等待外同步； SyncToGNSS1PPS_RisingEdge：可被系统中GNSS的1PPS上升沿连续同步； SyncToGNSS1PPS_FallingEdge：可被系统中GNSS的1PPS下降沿连续同步； SyncToGNSS1PPS_SingleRisingEdge：可被系统中GNSS的1PPS上升沿单次

	同步，调用IQS_SyncTimer_Single函数后开始等待外同步； SyncToGNSS1PPS_SingleFallingEdge：可被系统中GNSS的1PPS下降沿单次同步，调用IQS_SyncTimer_Single函数后开始等待外同步；。
double TriggerTimer_Period	设置定时触发周期，单位秒。仅在 TriggerSource = Timer 时生效。
uint8_t EnableReTrigger	使能设备在初始触发源触发后，接续进行自发的定时再触发。例如在每次外触发后，再以1ms间隔自动触发3次。仅在TriggerMode = FixedPoint 时生效。
double ReTrigger_Period	设置再触发的触发周期，单位s。
uint16_t ReTrigger_Count	设置每次触发后，再触发的执行次数。
DataFormat_TypeDef **DataFormat**	设置IQ数据的输出数据格式： Complex8bit：IQ两路数据为8位格式； Complex16bit：IQ两路数据为16位格式； Complex32bit：IQ两路数据为32位格式； Complexfloat：IQ，单路数据32位浮点（IQS模式不可用，仅由DDC函数输出数据时回写该枚举变量）。
GainStrategy_TypeDef **GainStrategy**	设置增益策略： LowNoisePreferred：侧重低噪声； HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef **Preamplifier**	设置前置放大器动作： AutoOn：自动使能前置放大器； ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。档位越高中频增益越高。
uint8_t EnableDebugMode	设置使能调试模式。厂商预留功能，请保持为0。
ReferenceClockSource_TypeDef **ReferenceClockSource**	设置参考时钟源： ReferenceClockSource_Internal：内部参考时钟（默认10MHz）； ReferenceClockSource_External：外部参考时钟（默认10MHz），当外部参考无法锁定时将自动切换为内部参考； ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：强制使用外部参考时钟，即使无法锁定也不会切换至内部参考。
double ReferenceClockFrequency	设置参考时钟频率 Hz。
SystemClockSource_TypeDef **SystemClockSource**	设置系统时钟源，默认使用内部系统时钟，请在厂商指导下使用： SystemClockSource_Internal: 内部系统时钟源；

	SystemClockSource_External: 外部系统时钟源。
Double ExternalSystemClockFrequency	设置外部系统时钟频率，Hz。
double NativeIQSampleRate_SPS	设置原生的IQ采样速率，对于可变采样率设备，可通过调整该参数对采样率进行调整；非可变采样率设备配总是配置为系统默认的固定值。（特定设备适用）
uint8_t EnableIFAGC	设置中频AGC控制： 0：AGC关闭，使用MGC方式； 1：AGC开启（特定设备适用）。
int8_t Atten	设置衰减dB,设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
DCCancelerMode_TypeDef DCCancelerMode	设置直流抑制器模式： DCCOff：直流抑制关闭； DCCHighPassFilterMode：高通滤波器模式（更好的抑制效果，但会抑制DC至100kHz范围内的信号）； DCCManualOffsetMode：开启，手动偏置模式（需要人工校准，但不低频损伤信号）。（特定设备适用） DCCAutoOffsetMode：开启，自动偏置方式。
QDCMode_TypeDef QDCMode	设置IQ幅相修正器模式： QDCOff: 关闭QDC功能； QDCManualMode：开启并使用手动模式； QDCAutoMode：开启并使用自动QDC模式。
float QDCIGain	设置归一化线性增益 I路，1.0 表示无增益，设置范围 0.8~1.2。QDCMode = QDCManualMode时生效。
float QDCQGain	设置归一化线性增益 Q路，1.0 表示无增益，设置范围 0.8~1.2。QDCMode = QDCManualMode时生效。
float QDCPhaseComp	设置相位补偿系数，设置范围 -0.2~+0.2。QDCMode = QDCManualMode 时生效。
int8_t DCCIOffset	设置I通道直流偏置，LSB。DCCancelerMode = DCCManualOffsetMode 时生效。
int8_t DCCQOffset	设置Q通道直流偏置，LSB。DCCancelerMode = DCCManualOffsetMode 时生效。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在Device_Open后调用。
示例	
int Status = -1;	

```
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
```

12.2 IQS_Configuration

int IQS_Configuration(void Device, const IQS_Profile_TypeDef* ProfileIn, IQS_Profile_TypeDef* ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)**

功能描述

此函数将频谱仪设备配置成IQ时域数据流模式(IQS)。IQS模式下，本振信号固定，并接收以本振频率为中心一定带宽的射频信号。同时在抽取倍数>2时可实现连续时域记录功能（需PC在某配置之上）。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
IQS_Profile_TypeDef *IQS_ProfileIn	IQS配置结构体指针，为输入变量。
IQS_Profile_TypeDef *IQS_ProfileOut	IQS配置结构体指针，为输出变量。
IQS_StreamInfo_TypeDef *StreamInfo	IQS时域数据流模式下，IQ数据流的相关信息。

IQS_StreamInfo_TypeDef 详细定义

double Bandwidth	信号带宽。
double IQSampleRate	IQ单路采样率，单位S/s（Sample/second）。
uint64_t PacketCount	单次触发的数据包总数。
uint64_t StreamSamples	单次触发的总数据点数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。
uint64_t StreamDataSize	单次触发的总数据字节数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。
uint32_t PacketSamples	单个数据包中包含的数据点数。即每次调用IQS_GetIQStream函数获取到的数据包中采样点数。

uint32_t PacketDataSize	单个数据包中包含的数据字节数。即每次调用IQS_GetIQStream函数所得到的数据字节数。
uint32_t GainParameter	增益参数信息。 Bit31~24：表示增益空间; Bit23~16：表示前置放大器状态； Bit15~0：表示频段信息。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_ProfileDeInit之后进行调用。

示例

```
int Status = -1; int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

12.3 IQS_BusTriggerStart

int IQS_BusTriggerStart(void Device)**

功能描述

此函数将发起总线触发。

兼容性	0.52.0及之后版本支持

参数说明

void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_GetIQStream之前进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
```

```
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef TraceInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
Status = IQS_BusTriggerStart(&Device);
```

12.4 IQS_BusTriggerStop

int IQS_BusTriggerStop(void** Device)

功能描述

此函数将终止当前总线触发。当配置TriggerMode = FixedPoints时，总线触发在由IQS_BusTriggerStart函数发起并达到指定触发长度后会自行终止，而无需调用该函数。

兼容性	0.52.0 及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非 0：异常，详见第六章与附录 1。
调用约束	需要在 IQS_GetIQStream 之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef TraceInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
```

Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);

Status = IQS_BusTriggerStop(&Device);

12.5 IQS_GetIQStream

IQS_GetIQStream(void Device, void** AlternIQStream, float* ScaleToV, IQS_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

调用此函数获取IQS模式下的时域数据、测量信息与触发信息

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
void AlternIQStream**	时域数据（交织IQ形式）的地址。一个数据包为固定64968个字节。当选择的IQ数据类型为int8_t时，I、Q两路分别为32484个点，每个点占1个字节；当选择的IQ数据类型为int16_t时，I、Q两路分别为16242个点，每个点占2个字节；当选择的IQ数据类型为int32_t时，I、Q两路分别为8121个点，每个点占4个字节。IQ数据是按照IQIQ…的排列方式存储。
Float* ScaleToV	时域数据至电压绝对值（V）的系数。
IQS_TriggerInfo_TypeDef *TriggerInfo	IQ数据流的触发相关信息。
MeasAuxInfo_TypeDef *MeasAuxInfo	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。
IQS_TriggerInfo_TypeDef 详细定义	
uint64_t SysTimerCountOfFirstDataPoint	首个数据点对应的系统时间计数器值。
uint16_t InPacketTriggeredDataSize	数据中有效触发数据的字节数。
uint16_t InPacketTriggerEdges	数据中所包含的边沿个数。
uint32_t StartDataIndexOfTriggerEdges[25]	各触发边沿的数据起始位置。
uint64_t SysTimerCountOfEdges[25]	各触发边沿的系统时间戳。
int8_t EdgeType[25]	各触发边沿的极性。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_Configuration之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
void* AlternIQStream = NULL;
float ScaleToV = 0;
IQS_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
IQS_GetIQStream(&Device, AlternIQStream,&ScaleToV,&TriggerInfo,&MeasAuxInfo);
```

13 IQ 数据记录 IQS 其他函数

13.1 IQS_MultiDevice_WaitExternalSync

int IQS_MultiDevice_WaitExternalSync(void Device, const IQS_Profile_TypeDef* ProfileIn)**

功能描述

调用该函数等待多几同步触发信号。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
const IQS_Profile_TypeDef *ProfileIn	IQS配置结构体指针，为输入变量。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_ Configuration之后进行调用，且TriggerSource选择MultiDevSyncByExt或MultiDevSyncByGNSS1PPS。

示例

```
int Status = -1; int DeviceNum0 = 0; int DeviceNum1 = 0;
void *Device0 = NULL; void *Device1 = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device0, DevNum0, &BootProfile, &BootInfo);
Status = Device_Open(&Device1, DevNum1, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn0, ProfileIn1;
IQS_Profile_TypeDef ProfileOut0, ProfileOut1;
IQS_TraceInfo_TypeDef StreamInfo0, StreamInfo1;
Status = IQS_ProfileDeInit(&Device0, &ProfileIn0);
Status = IQS_ProfileDeInit(&Device1, &ProfileIn1);
ProfileIn0.TriggerSource = MultiDevSyncByExt;
ProfileIn1.TriggerSource = MultiDevSyncByExt;
Status = IQS_Configuration(&Device0, &ProfileIn0, &ProfileOut0, &StreamInfo0);
Status = IQS_Configuration(&Device1, &ProfileIn1, &ProfileOut1, &StreamInfo1);
Status = IQS_MultiDevice_Run(&Device0);
Status = IQS_MultiDevice_Run(&Device1);
```

13.2 IQS_MultiDevice_Run

int IQS_MultiDevice_Run(void **Device)

功能描述

调用此函数使能多机同步运行。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_MultiDevice_WaitExternalSync之后进行调用。

示例

```
int Status = -1; int DeviceNum0 = 0; int DeviceNum1 = 0;
void *Device0 = NULL; void *Device1 = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device0, DevNum0, &BootProfile, &BootInfo);
Status = Device_Open(&Device1, DevNum1, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn0, ProfileIn1;
IQS_Profile_TypeDef ProfileOut0, ProfileOut1;
IQS_TraceInfo_TypeDef StreamInfo0, StreamInfo1;
Status = IQS_ProfileDeInit(&Device0, &ProfileIn0);
Status = IQS_ProfileDeInit(&Device1, &ProfileIn1);
ProfileIn0.TriggerSource = MultiDevSyncByExt;
ProfileIn1.TriggerSource = MultiDevSyncByExt;
Status = IQS_Configuration(&Device0, &ProfileIn0, &ProfileOut0, &StreamInfo0);
Status = IQS_Configuration(&Device1, &ProfileIn1, &ProfileOut1, &StreamInfo1);
Status = IQS_MultiDevice_Run(&Device0);
Status = IQS_MultiDevice_Run(&Device1);
```

13.3 IQS_SyncTimerByExtTrigger_Single

int IQS_SyncTimerByExtTrigger_Single(void Device)**

功能描述

调用此函数发起定时器-外触发单次同步。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_Configuration之后进行调用，且TriggerSource选择Timer。

示例

```
int Status = -1; int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_SyncTimerByExtTrigger_Single(&Device);
```

13.4 IQS_GetIQStream_PM1

int IQS_GetIQStream_PM1(void **Device, IQStream_TypeDef* IQStream)

功能描述

此函数获取IQS模式下的时域数据，时域数据格式为int8_t、int16_t和int32_t，根据用户需要自行选择数据格式。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
IQStream_TypeDef *IQStream	IQ数据流，包括IQ数据及相关配置信息等。
IQStream_TypeDef 详细定义	
void *AlternIQStream	返回时域IQ数据包。一整个数据包为固定64968个字节。当选择的IQ数据类型为int8_t时，I、Q两路分别为32484个点，每个点占1个字节；当选择的IQ数据类型为int16_t时，I、Q两路分别为16242个点，每个点占2

	个字节；当选择的IQ数据类型为int32_t时，I、Q两路分别为8121个点，每个点占4个字节。IQ数据是按照IQIQ…的排列方式存储。
float IQS_ScaleToV	时域数据至电压绝对值（V）的系数。
float MaxPower_dBm	数据中的功率最大值。
uint32_t MaxIndex	功率最大值在数据中的索引。
IQS_Profile_TypeDef IQS_Profile	数据的配置信息。
IQS_StreamInfo_TypeDef StreamInfo	数据的格式信息。
IQS_TriggerInfo_TypeDef TriggerInfo	数据的触发信息。
DeviceInfo_TypeDef DeviceInfo	数据的设备信息。
DeviceState_TypeDef DeviceState	数据的设备状态信息。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_Configuration之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
```

13.5 IQS_GetIQStream_Data

int IQS_GetIQStream_Data(void Device, int16_t IQ_data[])**

功能描述

调用此函数获取数据类型为int16_t的IQ时域数据。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	此参数设备句柄。
int16_t IQ_data[]	用于接收单路数据16位的IQ数据数组。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在IQS_Configuration之后进行调用。

示例

```
int Status = -1;int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
int16_t *IQ_data = new int16_t[StreamInfo.PacketSamples];
Status = IQS_GetIQStream_Data(&Device,IQ_data);
delete[] IQ_data;
```

14 检波分析 DET

DET为检波分析模式，对一定带宽内的信号进行功率检波，帮助用户观察信号的电平。

14.1 DET_ProfileDeInit

int DET_ProfileDeInit(void Device, DET_Profile_TypeDef* UserProfile_O)**

功能描述

此函数初始化配置DET模式的相关参数。DET模式下的中心频率、参考电平、抽取倍数等参数统一封装在DET_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
DET_Profile_TypeDef ***UserProfile_O**	DET配置结构体指针，为输入/输出变量。
DET_Profile_TypeDef 详细定义	
double CenterFreq_Hz	设置中心频率。
double RefLevel_dBm	设置参考电平。
uint32_t DecimateFactor	设置抽取倍数，有效分析带宽 = 原始分析带宽/抽取倍数。
RxPort_TypeDef **RxPort**	设置信号接收端口： ExternalPort：接收机接收来自外部输入端口输入的数据； InternalPort：接收机接收来自内部的辅助信号源的射频信号。
uint32_t BusTimeout_ms	设置数据传输超时时间（ms），如果设备在调用数据获取函数后于此时间内未能成功获取数据则函数返回错误。
DET_TriggerSource_TypeDef **TriggerSource**	与IQS_TriggerSource_TypeDef定义相同。
TriggerEdge_TypeDef **TriggerEdge**	与IQS模式中同名参数保持一致。
TriggerMode_TypeDef **TriggerMode**	
uint64_t TriggerLength	
TriggerOutMode_TypeDef **TriggerOutMode**	
TriggerOutPulsePolarity_TypeDef **TriggerOutPulsePolarity**	
double TriggerLevel_dBm	

double TriggerLevel_SafeTime	
double TriggerDelay	
double PreTriggerTime	
TriggerTimerSync_TypeDef	
TriggerTimerSync	
double TriggerTimer_Period	
uint8_t EnableReTrigger	
double ReTrigger_Period	
uint16_t ReTrigger_Count	
DETDetector_TypeDef	设置DET迹线检波器类型：
DET_TraceDetector	DET_Average：平均检波
	DET_Max：最大值检波
	DET_RMS：均方根值检波
	DET_Sample：取样检波
uint16_t DET_TraceDetectRatio	设置DET迹线检波比。
GainStrategy_TypeDef	设置增益策略：
GainStrategy	LowNoisePreferred：侧重低噪声；
	HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef	设置前置放大器动作：
Preamplifier	AutoOn：自动使能前置放大器；
	ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。档位越高中频增益越高。
uint8_t EnableDebugMode	设置使能调试模式。厂商预留功能，请保持为0。
ReferenceClockSource_TypeDef	与IQS模式中同名参数保持一致。
ReferenceClockSource	
double ReferenceClockFrequency	
SystemClockSource_TypeDef	
SystemClockSource	
Double	
ExternalSystemClockFrequency	
int8_t Atten	设置衰减dB,设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
DCCancelerMode_TypeDef	与IQS模式中同名参数保持一致。
DCCancelerMode	
QDCMode_TypeDef	

QDCMode

float QDCIGain

float QDCQGain

float QDCPhaseComp

int8_t DCCIOffset

int8_t DCCQOffset

返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在Device_Open后调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef UserProfile_O;
Status = DET_ProfileDeInit(&Device, &UserProfile_O);
```

14.2 DET_Configuration

int DET_Configuration(void Device, const DET_Profile_TypeDef* ProfileIn, DET_Profile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo)**

功能描述

此函数配置DET模式的相关参数。DET模式下的中心频率、参考电平、抽取倍数等参数统一封装在DET_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持

参数说明

void **Device	设备句柄。
DET_Profile_TypeDef *DET_ProfileIn	DET配置结构体指针，为输入变量。
DET_Profile_TypeDef *DET_ProfileOut	DET配置结构体指针，为输出变量。
DET_StreamInfo_TypeDef *StreamInfo	DET模式下，DET数据的相关信息。

DET_StreamInfo_TypeDef 详细定义

uint64_t PacketCount	单次触发的数据包总数。
uint64_t StreamSamples	单次触发的总数据点数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。
uint64_t StreamDataSize	单次触发的总数据字节数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。
uint64_t PacketSamples	单个数据包中包含的数据点数。即每次调用DET_GetPowerStream函数获取到的数据包中采样点数。
uint64_t PacketDataSize	单个数据包中包含的数据字节数。即每次调用DET_GetPowerStream函数所得到的数据字节数。
double TimeResolution	数据点的时间分辨率，单位秒。
uint32_t GainParameter	增益参数信息。 Bit31~24：表示增益空间; Bit23~16：表示前置放大器状态； Bit15~0：表示频段信息。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DET_ProfileDeInit之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

14.3 DET_BusTriggerStart

int DET_BusTriggerStart(void Device)**

功能描述

此函数将发起总线触发。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DET_GetPowerStream之前进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
```

14.4 DET_BusTriggerStop

int DET_BusTriggerStop(void Device)**

功能描述

此函数将终止当前总线触发。当配置TriggerMode = FixedPoints时，总线触发在由DET_BusTriggerStart函数发起并达到指定触发长度后会自行终止，而无需调用该函数。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DET_GetPowerStream之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
```

```
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
float NormalizedPowerStream;float ScaleToV;
float NormalizedPowerStream;float ScaleToV;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_GetPowerStream(&Device, &NormalizedPowerStream, & ScaleToV, &TriggerInfo, &MeasAuxInfo) ;
Status = DET_BusTriggerStop(&Device);
```

14.5 DET_GetPowerStream

int DET_GetPowerStream(void Device, float* NormalizedPowerStream, float* ScaleToV, DET_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

此函数，获取DET模式下的检波数据，并得到整型至绝对幅度(V单位)的比例因子及触发相关的信息，NormalizedPowerStream为I方+Q方开根号。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
float* NormalizedPowerStream	$\sqrt{I^2+Q^2}$的值。
float *ScaleToV	NormalizedPowerStream至电压绝对值（V）的系数。
DET_TriggerInfo_TypeDef *TriggerInfo	DET数据的触发相关信息。
MeasAuxInfo_TypeDef *MeasAuxInfo	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。

DET_TriggerInfo_TypeDef详细定义

uint64_t	同IQS_TriggerInfo_TypeDef
SysTimerCountOfFirstDataPoint	
uint16_t	
InPacketTriggeredDataSize	
uint16_t InPacketTriggerEdges	
uint32_t	
StartDataIndexOfTriggerEdges[25]	
uint64_t	
SysTimerCountOfEdges[25]	
int8_t EdgeType[25]	
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_BusTriggerStart(&Device);
float NormalizedPowerStream;float ScaleToV;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_GetPowerStream(&Device, &NormalizedPowerStream, & ScaleToV, &TriggerInfo, &MeasAuxInfo);
```

14.6 DET_SyncTimerByExtTrigger_Single

int DET_SyncTimerByExtTrigger_Single(void Device)**

功能描述

调用此函数发起定时器-外触发单次同步。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DET_Configuration之后进行调用，且TriggerSource选择Timer。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn;
DET_Profile_TypeDef ProfileOut;
DET_TraceInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_SyncTimerByExtTrigger_Single(&Device);
```

15 实时频谱 RTA

RTA为实时频谱分析模式，可帮助用户观察跳频或短暂的瞬态突发信号。

15.1 RTA_ProfileDeInit

int RTA_ProfileDeInit(void Device, RTA_Profile_TypeDef* UserProfile_O)**

功能描述

此函数初始化配置RTA模式的相关参数。RTA模式下的中心频率、参考电平、抽取倍数等参数统一封装在RTA_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
RTA_Profile_TypeDef *UserProfile_O	RTA配置结构体指针，为输入/输出变量。
RTA_Profile_TypeDef 详细定义	
double CenterFreq_Hz	设置中心频率。
double RefLevel_dBm	设置参考电平。
double RBW_Hz	设置RBW。
double VBW_Hz	设置VBW。
RBWMode_TypeDef RBW_Mode	设置RBW更新方式。手动输入、根据Span自动设置。
VBWMode_TypeDef VBW_Mode	设置VBW更新方式。手动输入、VBW = RBW、VBW = 0.1*RBW、 VBW = 0.01*RBW.
uint32_t DecimateFactor	设置抽取倍数，抽取倍数决定了RTA的带宽。
Window_TypeDef Window	设置FFT过程中使用的窗函数，：包含FlatTop、Blacnman_Nuttall、Blackman、Hamming和Hanning五个窗函数。
float FrameTimeMultiple	设置帧分析时间倍率，设备在单一频点上的分析时间 = 默认分析时间（系统自行设定） * 帧时间倍率。提高帧时间倍率将增加设备的最小扫描时间，但不严格线性。
FrameDetector_TypeDef FrameDetector	设置帧检波器类型： FrameDetector_Bypass：每个频点仅采样一帧数据，不进行帧间检波； FrameDetector_MaxHold：每个频点采样多帧，最终输出一帧，帧与帧取MaxHold； FrameDetector_Average：每个频点采样多帧，最终输出一帧；帧与帧取平均；

	FrameDetector_MinHold：每个频点的功率谱间进行帧检波，最终输出一帧，帧与帧取MinHold； FrameDetector_MaxPower：在FFT前，每个频点都进行长时间的采样，从中选取功率最大的帧数据进行FFT，用于捕获脉冲等瞬时信号（仅SWP模式可用）； FrameDetector_RawFrames：每个频点均进行多次采样，多次FFT分析，并逐帧输出功率谱（仅SWP模式可用）。
uint32_t TraceDetectRatio	设置视频检波比。
DetMode_TypeDef TraceDetector	设置视频检波过程中检波器类型：包含NegPeak,、PosPeak,、Sample、Normal、RMS和NoneDet六种检波方式。
RxPort_TypeDef RxPort	设置信号接收端口：，： ExternalPort：接收机接收来自外部输入端口输入的数据； InternalPort：接收机接收来自内部的辅助信号源的射频信号； ANT_Port、TR_Port、SWR_Port、INT_Port：only for TRx series。
uint32_t BusTimeout_ms	设置数据传输超时时间（ms），如果设备在调用数据获取函数后于此时间内未能成功获取数据则函数返回错误。
RTA_TriggerSource_TypeDef TriggerSource	与IQS_TriggerSource_TypeDef定义相同。
TriggerEdge_TypeDef TriggerEdge	设置触发边沿： RisingEdge：上升沿触发； FallingEdge：下降沿触发； DoubleEdge：双边沿触发。
TriggerMode_TypeDef TriggerMode	设置触发模式： FixedPoints：触发后，采集指定时间（TriggerAcqTime）的数据； Adaptive：触发后，持续采集数据，直到收到终止信号（外触发终止边沿或总线触发终止指令）。
double TriggerAcqTime	设置输入触发后的采样时间，仅在FixedPoints模式下生效。
TriggerOutMode_TypeDef TriggerOutMode	与IQS模式中同名参数保持一致。
TriggerOutPulsePolarity_TypeDef TriggerOutPulsePolarity	
double TriggerLevel_dBm	
double TriggerLevel_SafeTime	
double TriggerDelay	
double PreTriggerTime	
TriggerTimerSync_TypeDef	

TriggerTimerSync	
double TriggerTimer_Period	
uint8_t EnableReTrigger	
double ReTrigger_Period	
uint16_t ReTrigger_Count	
GainStrategy_TypeDef	设置增益策略：
GainStrategy	LowNoisePreferred：侧重低噪声； HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef	设置前置放大器动作：
Preamplifier	AutoOn：自动使能前置放大器； ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。档位越高中频增益越高。
uint8_t EnableDebugMode	设置使能调试模式。厂商预留功能，请保持为0。
ReferenceClockSource_TypeDef	与IQS模式中同名参数保持一致。
ReferenceClockSource	
double ReferenceClockFrequency	
SystemClockSource_TypeDef	
SystemClockSource	
double	
ExternalSystemClockFrequency	
int8_t Atten	设置衰减dB,设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
DCCancelerMode_TypeDef	与IQS模式中同名参数保持一致。
DCCancelerMode	
QDCMode_TypeDef	
QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在Device_Open后调用。
示例	

```
int Status = -1;
int DeviceNum = 0;
```

```
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef UserProfile_O;
Status = RTA_ProfileDeInit(&Device, &UserProfile_O);
```

15.2 RTA_Configuration

int RTA_Configuration(void Device, const RTA_Profile_TypeDef* ProfileIn, RTA_Profile_TypeDef* ProfileOut, RTA_FrameInfo_TypeDef* FrameInfo)**

功能描述

此函数配置RTA模式的相关参数。RTA模式下的中心频率、参考电平、抽取倍数等参数统一封装在RTA_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
RTA_Profile_TypeDef *RTA_ProfileIn	RTA配置结构体指针，为输入变量。
RTA_Profile_TypeDef *RTA_ProfileOut	RTA配置结构体指针，为输出变量。
RTA_FrameInfo_TypeDef *StreamInfo	RTA模式下，RTA数据的相关信息。

RTA_FrameInfo_TypeDef 详细定义

double StartFrequency_Hz	频谱的起始频率。
double StopFrequency_Hz	频谱的终止频率。
double POI	100%截获概率下的信号最短持续时间，以s为单位。
double TraceTimestampStep	每包数据内各条Trace的时间戳步进。(包整体时间戳为TriggerInfo中的SysTimerCountOfFirstDataPoint)。
double TimeResolution	每个时域数据的采样时间，也是时间戳的分辨率。
double PacketAcqTime	每包数据对应的采集时间。
uint32_t PacketCounts	总数据包数，仅在FixedPoints模式下生效。
uint32_t PacketFrames	每个数据包中的频谱帧数。
uint32_t FFTSize	频谱帧所采用的FFT点数。

uint32_t FrameWidth	频谱帧的频率点数，同概率密度图的X轴点数(宽度)。
uint32_t FrameHeight	频谱帧的幅度点数，同概率密度图的Y轴点数(高度)。
uint32_t PacketSamplePoints	每包数据对应的采集点数。
uint32_t PacketValidPoints	每包数据中所含的频域有效数据点数。
uint32_t MaxDensityValue	概率密度位图（BitMap）单个位点的值上限。
uint32_t GainParameter	增益参数信息。 Bit31~24：表示增益空间; Bit23~16：表示前置放大器状态； Bit15~0：表示频段信息。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
```

15.3 RTA_BusTriggerStart

int RTA_BusTriggerStart(void **Device)

功能描述

此函数将发起总线触发。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在RTA_GetRealTimeSpectrum之前进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_BusTriggerStart(&Device);
```

15.4 RTA_BusTriggerStop

int RTA_BusTriggerStop(void **Device)

功能描述

此函数将终止当前总线触发。当配置TriggerMode = FixedPoints时，总线触发在由RTA_BusTriggerStart函数发起并达到指定触发长度后会自行终止，而无需调用该函数。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在RTA_GetRealTimeSpectrum之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
```

```
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_BusTriggerStart(&Device);
uint8_t *SpectrumBitmap = new uint8_t[FrameInfo.PacketValidPoints];
uint16_t *SpectrumBitmap = new uint16_t[FrameInfo.FrameHeight * FrameInfo.FrameWidth];
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace, SpectrumBitmap,
&PlotInfo, &TriggerInfo,&MeasAuxInfo);
Status = RTA_BusTriggerStop(&Device);
delete[] SpectrumBitmap; delete[] SpectrumBitmap;
```

15.5 RTA_GetRealTimeSpectrum

int RTA_GetRealTimeSpectrum(void Device, uint8_t SpectrumStream[], uint16_t SpectrumBitmap[], RTA_PlotInfo_TypeDef* PlotInfo, RTA_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**

功能描述

此函数获取实时频谱模式下的频谱数据。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
uint8_t SpectrumStream[]	返回由连续频谱帧组成的频谱流，表示为相对功率，LSB = 0.75dB。此数组大小等于通过RTA_Configuration函数得到的RTA_FrameInfo.PacketValidPoints。
uint16_t SpectrumBitmap[]	返回概率密度图位图。此数组大小等于通过RTA_Configuration函数得到的RTA_FrameInfo.FrameHeight * RTA_FrameInfo.FrameWidth。
RTA_PlotInfo_TypeDef *RTA_PlotInfo	RTA获取后返回的绘图信息结构体。
RTA_TriggerInfo_TypeDef *TriggerInfo	RTA数据的触发相关信息。
MeasAuxInfo_TypeDef	返回测量数据的辅助信息，详见SWP_GetPartialSweep说明。

***MeasAuxInfo**	
RTA_PlotInfo_TypeDef 详细定义	
float ScaleTodBm	频谱帧幅度至dBm的比例因子与偏置值。频谱帧的幅度绝对功率等于
float OffsetTodBm	SpectrumStream[] * ScaleTodBm + OffsetTodBm。
uint64_t SpectrumBitmapIndex	当前概率密度图片段在完整触发数据中的序号。
RTA_TriggerInfo_TypeDef详细定义	
uint64_t SysTimerCountOfFirstDataPoint	与IQS模式中同名参数保持一致。
uint16_t InPacketTriggeredDataSize	
uint16_t InPacketTriggerEdges	
uint32_t StartDataIndexOfTriggerEdges[25]	
uint64_t SysTimerCountOfEdges[25]	
int8_t EdgeType[25]	
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在RTA_Configuration之后进行调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
uint8_t *SpectrumBitmap = new uint8_t[FrameInfo.PacketValidPoints];
uint16_t *SpectrumBitmap = new uint16_t[FrameInfo.FrameHeight * FrameInfo.FrameWidth];
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
```

```
Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace, SpectrumBitmap,
&PlotInfo, &TriggerInfo,&MeasAuxInfo);
delete[] SpectrumBitmap; delete[] SpectrumBitmap;
```

15.6 RTA_SyncTimerByExtTrigger_Single

int RTA_SyncTimerByExtTrigger_Single(void Device)**

功能描述

调用此函数发起定时器-外触发单次同步。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在RTA_Configuration之后进行调用，且TriggerSource选择Timer。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn;
RTA_Profile_TypeDef ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_SyncTimerByExtTrigger_Single(Device);
```

16 辅助信号源 ASG（选件）

ASG为辅助信号源功能，可以输出单音或扫频信号。

16.1 ASG_ProfileDeInit

int ASG_ProfileDeInit(void Device, ASG_Profile_TypeDef* Profile)**

功能描述

调用此函数初始化ASG功能的相关参数，初始化模拟信号源。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
ASG_Profile_TypeDef *Profile	ASG配置结构体指针。
ASG_Profile_TypeDef 详细定义	
double CenterFreq_Hz	中心频率，单位为Hz，在信号源工作在SIG_Fixed模式下生效；输入范围1M-1GHz;步进1Hz。
double Level_dBm	输出功率，单位为dBm，在信号源工作在SIG_Fixed模式下生效；输入范围-127--5dBm；步进0.25dB。
double StartFreq_Hz	频率扫描模式下的起始频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StopFreq_Hz	频率扫描模式下的终止频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StepFreq_Hz	频率扫描模式下的步进频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StartLevel_dBm	功率扫描模式下的起始功率，单位为Hz。
double StopLevel_dBm	功率扫描模式下的终止功率，单位为Hz。
double StepLevel_dBm	功率扫描模式下的步进功率，单位为Hz。
double DwellTime_s	频率扫描模式 或 功率扫描模式 下，单位为s，当触发模式为BUS时，扫描驻留时间，单位为s，在信号源工作在*Sweep*模式下生效；输入范围0-1000000；步进1。
double ReferenceClockFrequency	指定参考频率：对内参考和外参考均生效。
ReferenceClockSource_TypeDef ReferenceClockSource	选择参考时钟的输入源： ReferenceClockSource_Internal：内部时钟源(默认10MHz)； ReferenceClockSource_External：外部时钟源(默认10MHz)，当系统检测到外部时钟无法锁定时，将自动切换至内部参考；

	ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：外部时钟源，并且无视锁定情况，即使失锁也不会切换至内部参考。
ASG_Port_TypeDef **Port**	信号源输出端口： ASG_Port_External：外部端口。 ASG_Port_Internal：内部端口； ASG_Port_ANT：ANT端口（仅适用于TRx 系列）； ASG_Port_TR：TR端口（仅适用于TRx 系列）； ASG_Port_SWR：SWR端口（仅适用于TRx 系列）； ASG_Port_INT：INT端口（仅适用于TRx 系列）；
ASG_Mode_TypeDef **Mode**	关闭、点频、频率扫描（外触发，同步至接收）、功率扫描（外触发，同步至接收）： ASG_Mute：静音。 ASG_FixedPoint：定点； ASG_FrequencySweep：频率扫描； ASG_PowerSweep：功率扫描。
ASG_TriggerSource_TypeDef **TriggerSource**	信号源触发输入模式： ASG_TriggerIn_FreeRun：自由运行； ASG_TriggerIn_External：外触发； ASG_TriggerIn_Bus：定时器触发。
ASG_TriggerInMode_TypeDef **TriggerInMode**	信号源的触发模式： ASG_TriggerInMode_Null：自由运行； ASG_TriggerInMode_SinglePoint：单点触发（触发一次进行单次的频率或功率的配置）； ASG_TriggerInMode_SingleSweep：单次扫描触发（触发一次进行一个周期的扫描）； ASG_TriggerInMode_Continous：连续扫描触发（触发一次连续工作）。
ASG_TriggerOutMode_TypeDef **TriggerOutMode**	信号源的触发模式： ASG_TriggerOutMode_Null：自由运行。 ASG_TriggerOutMode_SinglePoint：单点触发（一次跳频输出一个脉冲）； ASG_TriggerOutMode_SingleSweep：单次扫描触发（一次扫描输出一个脉冲）。
返回值	0：无异常；非0：异常，详见第六章与附录1。

调用约束	需要Device_Open后调用。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
ASG_Profile_TypeDef *Profile;
Status= ASG_ProfileDeInit(&Device, Profile);
```

16.2 ASG_Configuration

int ASG_Configuration(void **Device, ASG_Profile_TypeDef *ProfileIn, ASG_Profile_TypeDef *ProfileOut,ASG_Info_TypeDef *ASG_Info)

功能描述

此函数配置ASG功能的相关参数。ASG模式下的中心频率、功率、驻留时间等参数统一封装在ASG_Profile_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **Device	设备句柄。
ASG_Profile_TypeDef *ProfileIn	ASG配置结构体指针，为输入变量。
ASG_Profile_TypeDef *ProfileOut	ASG配置结构体指针，为输出变量。
ASG_Info_TypeDef *ASG_Info	ASG模式下，ASG相关信息，为输出变量。
ASG_Info_TypeDef 详细定义	
uint32_t SweepPoints	扫描点数。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无。

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef *BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef *BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
ASG_Profile_TypeDef ProfileIn;
ASG_Profile_TypeDef ProfileOut;
ASG_Info_TypeDef ASG_Info;
ProfileIn. CenterFreq_Hz = 1e9;
Status = ASG_Configuration(&Device, &ProfileIn, &ProfileOut, &ASG_Info);
```

17 模拟信号解调 ASD

ADS为模拟调制解调处理接口，用户可调用其中的函数做一些模拟调制解调处理的相关操作。

17.1 ASD_Open

void ASD_Open (void **AnalogMod)

功能描述

此函数打开Analog功能，并在内存中分配一定空间以存储Analog的相关数据。在调用Analog的其它函数之前必须先调用此函数。当需要同时调用某一条函数时，可通过改变多加入几个Analog指针来进行操作。

兼容性	0.52.0及之后版本支持
参数说明	
void **Analog	运行Analog所需的内存空间引用。调用此函数后，函数将返回当前打开的Analog功能的内存地址。后续在调用其它API时，必须通过此引用来索引此次的地址。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在其它Analog函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的ASD_Open调用，必须在整个功能使用需求结束之后，调用ASD_Close函数，以释放内存。

示例

```
int Status = -1;
void *Analog = NULL;
Status = ASD_Open(&Analog);
```

17.2 ASD_Close

void ASD_Close (void AnalogMod)**

功能描述

此函数关闭Analog功能，释放所开辟的内存空间。

兼容性	0.52.0及之后版本支持
参数说明	
void **Analog	运行Analog所需的内存空间引用。

返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后Analog功能关闭，内存空间释放。如需要重新调用Analog功能，则再次通过调用ASD_Open，打开Analog功能。

示例

```
int Status = -1; void *Analog = NULL;
Status = ASD_Open(&Analog);
Status = ASD_Close(&Analog);
```

17.3 ASD_Demodulate_FM

int ASD_Demodulate_FM(void AnalogMod, const IQStream_TypeDef* IQStreamIn, bool reset, float result[], double* carrierOffsetHz)**

功能描述

此函数对IQ数据进行FM解调。

兼容性	0.52.0及之后版本支持
参数说明	
void **Analog	运行Analog所需的内存空间引用。
IQStream_TypeDef *IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。
bool reset	重置缓存，对于一次连接解调来说，只有第一次调用函数的时候需要置成1，后面都是置0。
float result[]	FM解调后的信号数据，数据长度与IQ信号数据长度相同。
double carrierOffsetHz	载波的频率偏移。
调用约束	需要在ASD_Open之后进行调用。

示例

```
int Status = -1;
void *Analog = NULL;
Status = ASD_Open(&Analog);
bool reset = 1;
double carrierOffsetHz = 0;
float *result = new float[IQStreamIn.StreamInfo.PacketSamples];
Status = ASD_Demodulate_FM(&Analog, &IQStream, reset, result, &carrierOffsetHz);
delete[] result;
Status = ASD_Close(&Analog);
```

17.4 ASD_Demodulate_AM

int ASD_Demodulate_AM(const IQStream_TypeDef* IQStreamIn, float result[])

功能描述

此函数对IQ数据进行AM解调。

兼容性	0.52.0及之后版本支持
参数说明	
void **Analog	运行Analog所需的内存空间引用。
IQStream_TypeDef *IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。
float result[]	AM解调后的信号数据，数据长度与IQ信号数据长度相同。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在ASD_Open之后进行调用。

示例

```
int Status = -1;
void *Analog = NULL;
Status = ASD_Open(&Analog);
bool reset = 1;
double carrierOffsetHz = 0;
float *result = new float[IQStreamIn.StreamInfo.PacketSamples];
Status = ASD_Demodulate_AM(&Analog, &IQStream，result);
delete[] result;
Status = ASD_Close(&Analog);
```

18 数字信号处理 DSP 迹线分析

18.1 DSP_TraceAnalysis_IM3

int DSP_TraceAnalysis_IM3(const double Freqs[], const float PowerSpec_dBm[], uint32_t TracePoints, TraceAnalysisResult_IP3_TypeDef* IM3Result)

功能描述

此函数分析迹线的 IM3 参数。

兼容性	0.52.0及之后版本支持
参数说明	
double Freqs[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
TraceAnalysisResult_IP3_TypeDef *IM3Result	返回IP3的测量结果。
TraceAnalysisResult_IP3_TypeDef 详细定义	
double LowToneFreq	低音信号频率，单位随数据源。
double HighToneFreq	高音信号频率，单位随数据源。
double LowIM3PFreq	低频交调频率，单位随数据源。
double HighIM3PFreq	高频交调频率，单位随数据源。
float LowTonePower_dBm	低音功率, dBm。
float HighTonePower_dBm	高音功率, dBm。
float TonePowerDiff_dB	低音功率 - 高音功率。
float LowIM3P_dBc	LowIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - LowTonePower_dBm, 低频交调产物相对于最强主信号的强度。
float HighIM3P_dBc	HighIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - HighTonePower_dBm, 高频交调强产物相对于最强主信号的度。
float IP3_dBm	IP3的测试结果。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
```

```
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
TraceAnalysisResult_IP3_TypeDef IM3Result;
Status = DSP_TraceAnalysis_IM3(Frequency, PowerSpec_dBm, TraceInfo.FullsweepTracePoints, &IM3Result);
delete[] Frequency;delete[] PowerSpec_dBm;
```

18.2 DSP_TraceAnalysis_IM2

int DSP_TraceAnalysis_IM2(const double Freqs[], const float PowerSpec_dBm[], uint32_t TracePoints, TraceAnalysisResult_IP2_TypeDef* IM2Result)

功能描述

此函数分析迹线的 IM2 参数。

兼容性	0.52.0及之后版本支持
参数说明	
double Freqs[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
TraceAnalysisResult_IP2_TypeDef *IM2Result	返回IP2的测量结果。

TraceAnalysisResult_IP2_TypeDef 详细定义

double LowToneFreq	低音信号频率，单位随数据源。
double HighToneFreq	高音信号频率，单位随数据源。
double IM2PFreq	低频交调频率，单位随数据源。
float LowTonePower_dBm	低音功率, dBm。
float HighTonePower_dBm	高音功率, dBm。

float TonePowerDiff_dB	低音功率 - 高音功率。
float IM2P_dBc	IM2P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - IM2P_dBm, 低频交调产物相对于最强主信号的强度。
float IP2_dBm	IP2的测试结果。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
TraceAnalysisResult_IP2_TypeDef IM2Result;
Status = DSP_TraceAnalysis_IM2(Frequency, PowerSpec_dBm, TraceInfo.FullsweepTracePoints, &IM2Result);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.3 DSP_TraceAnalysis_PhaseNoise

int DSP_TraceAnalysis_PhaseNoise(const double Freq_Hz[], const float PowerSpec_dBm[], const double* OffsetFreqs, uint32_t TracePoints, uint32_t OffsetFreqsToAnalysis, double* CarrierFreqOut, float* PhaseNoiseOut_dBc)

功能描述

此函数分析迹线的 相位噪声参数。

兼容性	0.52.0及之后版本支持
参数说明	
double Freq_Hz[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
double* OffsetFreqs	频率偏移数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
uint32_t OffsetFreqsToAnalysis	需要分析的频偏频点数。
double* CarrierFreqOut	实际的载波频率。
float* PhaseNoiseOut_dBc	频偏数组对应的相位噪声。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
int32_t OffsetFreqsToAnalysis = 3;
double *OffsetFreqs = new double[OffsetFreqsToAnalysis];
OffsetFreqs[0  = 1e6;
OffsetFreqs[1] = 100e3;
OffsetFreqs[2] = 10e3;
double CarrierFreqOut = 0;
float *PhaseNoiseOut_dBc = new float[OffsetFreqsToAnalysis];
```

```
Status=DSP_TraceAnalysis_PhaseNoise(Frequency,PowerSpec_dBm,OffsetFreqs,
TraceInfo.FullsweepTracePoints,OffsetFreqsToAnalysis, &CarrierFreqOut,PhaseNoiseOut_dBc);
delete[] Frequency;
delete[] PowerSpec_dBm;
delete[] OffsetFreqs ;
delete[] PhaseNoiseOut_dBc;
```

18.4 DSP_TraceAnalysis_ChannelPower

int DSP_TraceAnalysis_ChannelPower(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const double CenterFrequency, const double AnalysisSpan, const double RBW, DSP_ChannelPowerInfo_TypeDef* ChannelPowerResult)

功能描述

此函数分析迹线的信道功率。

兼容性	0.52.0及之后版本支持
参数说明	
double Freq_Hz[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
double CenterFrequency	需要测量的信道中心频率。
double AnalysisSpan	需要测量的信道带宽。
double RBW	分辨率带宽。
DSP_ChannelPowerInfo_TypeDef *ChannelPowerResult	返回信道功率的测量结果。
DSP_ChannelPowerInfo_TypeDef	信道功率结构体
详细定义	
float ChannelPower_dBm	信道功率(dBm)
float PowerDensity	信道功率密度(dBm/Hz)
float ChannelPeakIndex	信道内的峰值索引
double ChannelPeakFreq_Hz	信道内的峰值频率(Hz）
float ChannelPeakPower_dBm	信道内的峰值功率(dBm）
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
```

```
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float * PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
double CenterFrequency = 1e9;
double AnalysisSpan = 50e6;
DSP_ChannelPowerInfo_TypeDef ChannelPowerResult;
Status=DSP_TraceAnalysis_ChannelPower(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,CenterFrequency,
AnalysisSpan,&ChannelPowerResult);
delete[] Frequency;delete[] PowerSpec_dBm;
```

18.5 DSP_TraceAnalysis_XdBBW

int DSP_TraceAnalysis_XdBBW(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const float XdB, TraceAnalysisResult_XdB_TypeDef* XdBResult)

功能描述

此函数分析迹线的 XdB带宽。

兼容性	0.52.0及之后版本支持
参数说明	
double Freq_Hz[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
float XdB	峰值功率需要下降的dB。
TraceAnalysisResult_XdB_TypeDef* XdBResult	返回XdB的测量结果。

TraceAnalysisResult_XdB_TypeDef 详细定义	XdB带宽信息结构体
double XdBBandWidth_Hz	XdB带宽(Hz)。
double CenterFreq_Hz	XdB带宽的中心频率(Hz)。
double StartFreq_Hz	XdB带宽的起始频率(Hz)。
double StopFreq_Hz	XdB带宽的终止频率(Hz)。
float StartPower_dBm	XdB带宽的起始频率对应的功率(dBm)。
float StopPower_dBm	XdB带宽的终止频率对应的功率(dBm)。
uint32_t PeakIndex	XdB带宽内的峰值索引。
double PeakFreq_Hz	XdB带宽内的峰值频率(Hz）。
float PeakPower_dBm	XdB带宽内的峰值功率(dBm)。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
float XdB = 3;
TraceAnalysisResult_XdB_TypeDef XdBResult
Status=DSP_TraceAnalysis_XdBBW(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,XdB,
&XdBResult);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.6 DSP_TraceAnalysis_OBW

int DSP_TraceAnalysis_OBW(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const float OccupiedRatio, TraceAnalysisResult_OBW_TypeDef* OBWResult)

功能描述

此函数分析迹线的 占用带宽。

兼容性	0.52.0及之后版本支持
参数说明	
double Freq_Hz[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
float OccupiedRatio	需要测试的占用带宽比例，通常情况下设置为0.99。
TraceAnalysisResult_OBW_Type Def* OBWResult	返回占用带宽的测量结果。
TraceAnalysisResult_OBW_Type Def 详细定义	
double OccupiedBandWidth	占用带宽(Hz)。
double CenterFreq_Hz	占用带宽的中心频率(Hz)。
double StartFreq_Hz	占用带宽的起始频率(Hz)。
double StopFreq_Hz	占用带宽的终止频率(Hz)。
float StartPower_dBm	占用带宽的起始频率对应的功率(dBm)。
float StopPower_dBm	占用带宽的终止频率对应的功率(dBm)。
float StartRatio	占用带宽的起始频率对应的功率占比。
float StopRatio	占用带宽的终止频率对应的功率占比。.
uint32_t PeakIndex	占用带宽内的峰值索引
double PeakFreq_Hz	占用带宽内的峰值频率(Hz）。
float PeakPower_dBm	占用带宽内的峰值功率(dBm)。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
float OccupiedRatio = 0.99;
TraceAnalysisResult_OBW_TypeDef OBWResult;
Status=DSP_TraceAnalysis_OBW(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,OccupiedRatio,
&OBWResult);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

18.7 DSP_TraceAnalysis_ACPR

int DSP_TraceAnalysis_ACPR(const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo, TraceAnalysisResult_ACPR_TypeDef* ACPRResult)

功能描述

此函数分析迹线的 邻道功率比。

兼容性	0.52.0及之后版本支持
参数说明	
double Freq_Hz[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo	测试邻道功率比需要给定的变量。
TraceAnalysisResult_ACPR_TypeDef* ACPRResult	返回邻道功率比的测量结果。
DSP_ACPRFreqInfo_TypeDef 详细	邻道功率比频率信息结构体

定义

double RBW	分辨率带宽(Hz)。
double MainChCenterFreq_Hz	主信道中心频率(Hz)。
double MainChBW_Hz	主信道带宽(Hz)。
double AdjChSpace_Hz	邻道间隔，主信道中心频率与邻道中心频率的差值(Hz)。
uint32_t AdjChPair	邻道对。1代表左右2个邻道，2代表左右4个邻道。

TraceAnalysisResult_ACPR_TypeDef 详细定义

float MainChPower_dBm	主信道功率(dBm)。
uint32_t MainChPeakIndex	主信道的峰值索引。
double MainChPeakFreq_Hz	主信道的峰值频率(Hz）。
float MainChPeakPower_dBm	主信道峰值功率(dBm)。
double L_AdjChCenterFreq_Hz	左邻道中心频率(Hz)。
double L_AdjChBW_Hz	左邻道带宽(Hz)。
float L_AdjChPower_dBm	左邻道功率(dBm)。
float L_AdjChPowerRatio	左邻道功率比（左邻道功率/主信道功率）。
float L_AdjChPowerDiff_dBc	左邻道功率差（左邻道功率-主信道功率 dBc）。
float L_AdjChPeakIndex	左邻道的峰值索引。
double L_AdjChPeakFreq_Hz	左信道的峰值频率(Hz）。
float L_AdjChPeakPower_dBm	左邻道的值功率(dBm)。
double R_AdjChCenterFreq_Hz	右邻道中心频率(Hz)。
double R_AdjChBW_Hz	右邻道带宽(Hz)。
float R_AdjChPower_dBm	右邻道功率(dBm)。
float R_AdjChPowerRatio	右邻道功率比（右邻道功率/主信道功率）。
float R_AdjChPowerDiff_dBc	右邻道功率差（右邻道功率-主信道功率 dBc）。
float R_AdjChPeakIndex	右邻道的峰值索引。
double R_AdjChPeakFreq_Hz	右信道的峰值频率(Hz）。
float R_AdjChPeakPower_dBm	右邻道的值功率(dBm)。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
double *Frequency = new double[TraceInfo.FullsweepTracePoints];
float* PowerSpec_dBm = new float[TraceInfo.FullsweepTracePoints];
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency, PowerSpec_dBm,&MeasAuxInfo);
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo;
ACPRFreqInfo.RBW = ProfileOut.RBW_Hz;
ACPRFreqInfo.MainChCenterFreq_Hz = 1e9;
ACPRFreqInfo.MainChBW_Hz = 50e6;
ACPRFreqInfo.AdjChSpace_Hz = 50e6;
ACPRFreqInfo.AdjChPair = 1;
TraceAnalysisResult_ACPR_TypeDef ACPRPowerInfo;
Status=DSP_TraceAnalysis_ACPR(Frequency,PowerSpec_dBm,TraceInfo.FullsweepTracePoints,ACPRFreqInfo,
&ACPRPowerInfo);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

19 数字信号处理 DSP 流数据的分析与处理

19.1 DSP_Open

int DSP_Open(void DSP)**

功能描述

此函数打开DSP功能，并在内存中分配一定空间以存储DSP的相关数据。在调用DSP的其它函数之前必须先调用此函数。当需要同时调用某一条函数时，可通过改变多加入几个DSP指针来进行操作。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。调用此函数后，函数将返回当前打开的DSP功能的内存地址。后续在调用其它API时，必须通过此引用来索引此次的地址。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	必须在其它DSP函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的DSP_Open调用，必须在整个功能使用需求结束之后，调用DSP_Close函数，以释放内存。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
```

19.2 DSP_Close

int DSP_Close(void DSP)**

功能描述

此函数关闭DSP功能，释放所开辟的内存空间。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后DSP功能关闭，内存空间释放，如需要重新调用DSP功能。则需要再次通过调用DSP_Open，

打开DSP功能。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
Status = DSP_Close(&DSP);
```

19.3 DSP_FFT_DeInit

int DSP_FFT_DeInit(DSP_FFT_TypeDef* IQToSpectrum)

功能描述

此函数初始化配置FFT模式的相关参数。FFT模式下的FFT点数、窗型、抽取倍数等参数统一封装在DSP_FFT_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
DSP_FFT_TypeDef *IQToSpectrum	DSP_FFT配置结构体指针，为输入/输出变量。
DSP_FFT_TypeDef 详细定义	
uint32_t FFTSize	FFT分析点数。
uint32_t SamplePts	有效采样点数。
Window_TypeDef Window	设置FFT过程中使用的窗函数： FlatTop：平顶窗； Blackman_Nuttall：Nuttall窗； Blackman：布莱克曼窗；（暂未开放） Hamming：汉明窗；（暂未开放） Hanning：汉宁窗。（暂未开放）
DetMode_TypeDef Detector	设置视频检波过程中检波器类型： NegPeak：负峰值检波； Sample：采样检波； Normal：Normal检波； PosPeak：峰值检波； RMS：均方根检波； NoneDet：不进行视频检波。
uint32_t DetectionRatio	迹线检波比。
float Intercept	设置输出频谱截取，例如 Intercept = 0.8 则表示输出 80%的频谱结果。

bool Calibration	设置是否进行校准。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_FFT_Configuration前调用。

示例

```
int Status = -1;
DSP_FFT_TypeDef UserProfile_O;
Status = DSP_FFT_DeInit(&UserProfile_O);
```

19.4 DSP_FFT_Configuration

int DSP_FFT_Configuration(void DSP, const DSP_FFT_TypeDef* ProfileIn, DSP_FFT_TypeDef* ProfileOut, uint32_t* TracePoints, double* RBWRatio)**

功能描述

此函数配置FFT模式的相关参数。FFT模式下的FFT点数、窗型、抽取倍数等参数统一封装在DSP_FFT_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
DSP_FFT_TypeDef *ProfileIn	DSP_FFT配置结构体指针，为输入变量。
DSP_FFT_TypeDef *ProfileOut	DSP_FFT配置结构体指针，为输出变量。
uint32_t TracePoints	当前DSP_FFT配置下可获取到的频谱点数。
double *RBWRatio	返回RBW相对于采样率的比值。RBW = RBWRatio * IQSampleRate
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DSP_FFT_DeInit之后进行调用。

示例

```
int Status = -1; void *DSP = NULL; uint32_t TracePoints = 0; DSP_FFT_TypeDef ProfileIn,ProfileOut;
Status = DSP_Open(&DSP);
Status = DSP_FFT_DeInit(&ProfileIn);
Status = DSP_FFT_Configuration(&DSP, &ProfileIn, &ProfileOut, &TracePoints);
```

19.5 DSP_FFT_IQSToSpectrum

int DSP_FFT_IQSToSpectrum(void DSP_FFT, const IQStream_TypeDef* IQStream, double* Frequency, float* PowerSpec_dBm)**

功能描述

此函数将IQ数据转换成频谱数据，包括频率和功率。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
IQStream_TypeDef *IQStream	IQ数据流的相关信息，包括IQ数据及相关配置信息。
double *Frequency	返回频谱数据的频率数组。数组大小为TracePoints，由DSP_FFT_Configuration()函数输出得到。
float *PowerSpec_dBm	返回频谱数据的功率数组。数组大小为TracePoints，由DSP_FFT_Configuration()函数输出得到。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DSP_FFT_Configuration之后进行调用。

示例

```
int Status = -1;void *DSP = NULL;Status = DSP_Open(&DSP); uint32_t TracePoints = 0;
DSP_FFT_TypeDef ProfileIn,ProfileOut;
Status = DSP_FFT_DeInit(&ProfileIn);
Status = DSP_FFT_Configuration(&DSP, &ProfileIn, &ProfileOut, &TracePoints);
double *Frequency = new double[TracePoints];
float * PowerSpec_dBm = new float[TracePoints];
Status = DSP_FFT_IQSToSpectrum(&Device, &IQStream, Frequency, PowerSpec_dBm);
delete[] Frequency;
delete[] PowerSpec_dBm;
```

19.6 DSP_DDC_DeInit

int DSP_DDC_DeInit(DSP_DDC_TypeDef* DDC_ProfileIn)

功能描述

此函数初始化DDC模式的相关参数。DDC模式下的复混频和重采样参数统一封装在DSP_DDC_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
DSP_DDC_TypeDef *DDC_ProfileIn	DSP_DDC配置结构体指针，为输入/输出变量。
DSP_DDC_TypeDef 详细定义	
double DDCOffsetFrequency	设置复混频的频率偏移值。
double SampleRate	设置复混频的采样率，需要与IQ数据采样率相同。
float DecimateFactor	设置重采样抽取倍数，范围:1 ~ 2^16。
uint64_t SamplePoints	设置复混频的采样点数。

返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_DDC_Configuration前调用。

示例

```
int Status = -1;
DSP_DDC_TypeDef DDC_ProfileIn;
Status = DSP_DDC_DeInit(&DDC_ProfileIn);
```

19.7 DSP_DDC_Configuration

int DSP_DDC_Configuration(void DSP, const DSP_DDC_TypeDef* DDC_ProfileIn, DSP_DDC_TypeDef* DDC_ProfileOut)**

功能描述

此函数配置DDC模式的相关参数。DDC模式下的复混频和重采样参数统一封装在DSP_DDC_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
DSP_DDC_TypeDef *ProfileIn	DSP_DDC配置结构体指针，为输入变量。
DSP_DDC_TypeDef *ProfileOut	DSP_DDC配置结构体指针，为输出变量。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DSP_DDC_DeInit之后进行调用。

示例

```
int Status = -1;void *DSP = NULL;Status = DSP_Open(&DSP);
DSP_DDC_TypeDef ProfileIn,ProfileOut;
Status = DSP_DDC_DeInit(&ProfileIn);
uint32_t DDC_Points = 0;
Status = DSP_DDC_Configuration(&DSP, &ProfileIn, &ProfileOut, &DDC_Points);
```

19.8 DSP_DDC_Reset

void DSP_DDC_Reset(void DSP)**

功能描述

此函数重置DDC中的缓存。

兼容性	0.52.0及之后版本支持
参数说明	

void **DSP	运行DSP所需的内存空间引用。
返回值	无返回值。
调用约束	需要在DSP_Open之后进行调用。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
Status = DSP_DDC_Reset(&DSP);
```

19.9 DSP_DDC_Excute

int DSP_DDC_Excute(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**

功能描述

此函数将IQ数据进行数字下变频。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
IQStream_TypeDef *IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。
IQStream_TypeDef *IQStreamOut	输出IQ数据流的相关信息，包括IQ数据及相关配置信息。
调用约束	需要在DSP_DDC_Configuration之后进行调用。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
DSP_DDC_TypeDef ProfileIn,ProfileOut;
Status = DSP_DDC_DeInit(&ProfileIn);
uint32_t DDC_Points = 0;
Status = DSP_DDC_Configuration(&DSP, &ProfileIn, &ProfileOut);
IQStream_TypeDef IQStreamOut;
Status = DSP_DDC_Excute(&DSP, &IQStreamIn, &IQStreamOut);
```

19.10 DSP_AudioAnalysis

void DSP_AudioAnalysis(const double Audio[], const uint64_t SamplePoints, const double

SampleRate, DSP_AudioAnalysis_TypeDef* AudioAnalysis)

功能描述

此函数分析音频的 音频电压(V)、音频频率(Hz)、信纳德(dB)和总谐波失真(%) 参数。

兼容性	0.52.0及之后版本支持

参数说明

double Audio[]	音频信号数组。
uint64_t SamplePoints	音频信号数组的长度。
double SampleRate	音频信号的采样率。
DSP_AudioAnalysis_TypeDef* AudioAnalysis	返回音频分析的测量结果。

DSP_AudioAnalysis_TypeDef 详细定义

double AudioVoltage	音频电压(V)。
double AudioFrequency	音频频率(Hz)。
double SINDA	信纳德(dB)。
double THD	总谐波失真(%)。
返回值	无返回值。
调用约束	无

示例

```
int Status = -1;
int DeviceNum = 0;
void *Device = NULL;
BootProfile_TypeDef BootProfile_IO;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_TraceInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device,&ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
void* AlternIQStream = NULL;
float ScaleToV = 0;
IQS_TriggerInfo_TypeDef TriggerInfo;
int32_t  I_MaxValue = 0;
```

```
uint32_t  I_MaxIndex=0;
IQS_GetIQStream(&Device, AlternIQStream,&ScaleToV,&TriggerInfo,&I_MaxValue,&I_MaxIndex);
double*Audio= new double[StreamInfo.PacketSamples];
int16_t* IQ = (int16_t*)AlternIQStream;
for(uint64_t i = 0;i<StreamInfo.PacketSamples;i++){
    Audio[i] = ((double)IQ[2*i])*ScaleToV ;
}
DSP_AudioAnalysis_TypeDef AudioAnalysis;
DSP_AudioAnalysis(Audio,StreamInfo.PacketSamples,StreamInfo.IQSampleRate,&AudioAnalysis);
delete[] Audio;
```

19.11 DSP_LPF_DeInit

void DSP_LPF_DeInit(Filter_TypeDef* LPF_ProfileIn)

功能描述

此函数初始化LPF模式的相关参数。LPF模式下的滤波器抽头数、截止频率、阻带衰减等参数统一封装在Filter_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
Filter_TypeDef *LPF_ProfileIn	Filter_TypeDef配置结构体指针，为输入/输出变量。
Filter_TypeDef 详细定义	低通滤波器结构体
int n	设置滤波器抽头数（ n > 0 ）。
float fc	设置截止频率 (截止频率/采样率 0 < fc < 0.5)。
float As	设置阻带衰减（ As > 0 , [dB]）。
float mu	设置分数采样偏移量（ -0.5 < mu < 0.5 ）。
uint32_t SamplePts	设置采样点数（Samples > 0）。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_LPF_Configuration前调用。

示例

```
Filter_TypeDef LPF_ProfileIn;
DSP_LPF_DeInit(&LPF_ProfileIn);
```

19.12 DSP_LPF_Configuration

void DSP_LPF_Configuration(void DSP, const Filter_TypeDef* LPF_ProfileIn, Filter_TypeDef***

LPF_ProfileOut)

功能描述

此函数配置LPF模式的相关参数。LPF模式下的滤波器抽头数、截止频率、阻带衰减等参数统一封装在Filter_TypeDef结构体中。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
Filter_TypeDef *LPF_ProfileIn	Filter_TypeDef配置结构体指针，为输入/输出变量。
Filter_TypeDef *LPF_ProfileOut	Filter_TypeDef配置结构体指针，为输入/输出变量。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	需要在DSP_LPF_DeInit之后进行调用。

示例

```
int Status = -1;void *DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn;
Filter_TypeDef LPF_ProfileOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
```

19.13 DSP_LPF_Reset

void DSP_LPF_Reset(void DSP)**

功能描述

此函数重置LPF中的缓存。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_Open后调用。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
DSP_LPF_Reset(&DSP);
```

19.14 DSP_LPF_Execute_Real

void DSP_LPF_Execute_Real(void DSP, float* Signal, float* LPF_Signal)**

功能描述

此函数对实信号进行低通滤波。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
float *Signal	输入的信号。
float *LPF_Signal	经过低通滤波后的信号。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_LPF_Configuration后调用。

示例

```
int Status = -1;
void *DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn;
Filter_TypeDef LPF_ProfileOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
LPF_ProfileIn.SamplePts = 16242;
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
float* LPF_Signal = new float[LPF_ProfileIn.SamplePts];
DSP_LPF_Execute_Real(&DSP, Signal, LPF_Signal);
delete[] LPF_Signal;
```

19.15 DSP_LPF_Execute_Complex

void DSP_LPF_Execute_Complex(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**

功能描述

此函数对IQ信号进行低通滤波。

兼容性	0.52.0及之后版本支持
参数说明	
void **DSP	运行DSP所需的内存空间引用。
IQStream_TypeDef *IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。

IQStream_TypeDef *IQStreamOut	输出IQ数据流的相关信息，包括IQ数据及相关配置信息。
返回值	0：无异常；非0：异常，详见第六章与附录1。
调用约束	在DSP_LPF_Configuration后调用。
示例	

```
int Status = -1; void *DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn,LPF_ProfileOut; IQStream_TypeDef IQStreamOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
DSP_LPF_Execute_Complex(&DSP, &IQStreamIn, &IQStreamOut);
```

20 附录 Appendix 1：API 返回值索引

表9　API 返回值说明

错误代码	错误/警告原因	类型[1]	处理
0	无错误	-	无需处理，可正常执行后续过程。
-1	总线打开错误	错误	检查设备供电、数据线连接，检查驱动程序是否正确安装。排除错误后需要重新调用 Device_Open 以打开设备。
-3	射频校准文件丢失[2]	错误	检查射频校准文件是否放置在规定的目录下。排除错误后需要重新调用 Device_Open 打开设备。
-4	中频校准文件丢失[2]	错误	检查中频校准文件是否放置在规定的目录下。排除错误后需要重新调用 Device_Open 打开设备。
-5	设备配置信息丢失[2]	错误	检查所使用的射频校准文件与中频校准文件是否正确。排除错误后需要重新调用 Device_Open 打开设备。
-6	设备规格文件丢失[2]	错误	检查设备规格文件（若需要）是否放置在规定的目录下
-7	更新 Strategy 失败	错误	重新调用 Device_Open 打开设备。
-8	总线通信错误	错误	重新调用当前模式下的配置函数。
-9	数据内容错误	错误	重新调用当前模式下的配置函数。
-10	未在指定时间内获取到数据	警告	检查触发源是否正常输出触发信号，若无异常，可继续调用当前函数直到取得数据。
-11	通过总线下发配置错误	警告	重新调用 Configuration 函数配置设备。
-12	输入信号幅度超过当前配置下的额定量程	警告	当前函数获取降低输入信号幅度或适当增大参考电平。
-14	距离最近一次配置，温度产生较大变化	警告	设备温度距离上一次配置出现较大变化，建议重新调用 Configuration 函数配置设备，以获得最佳性能。
-15	本振或时钟存在锁定异常	警告	本振或时钟存在锁定可能异常，建议重新调用 Configuration 函数配置设备，尝试恢复正常状态。

[1] 类型为“错误”时，需要立即排除问题，并重新打开设备，否则设备无法继续运行后续流程。类型为“警告”时，设备可继续后续流程，无需关闭或重新打开设备，但仍然建议针对具体的返回值与当前的应用场景选择性的处理。

[2] 对于返回值为-3、-4、-5、-6 的情况，还需要确认文件存放路径是否为全英文路径。若路径中包含中文字符，调用 API 时也会提示文件加载失败。