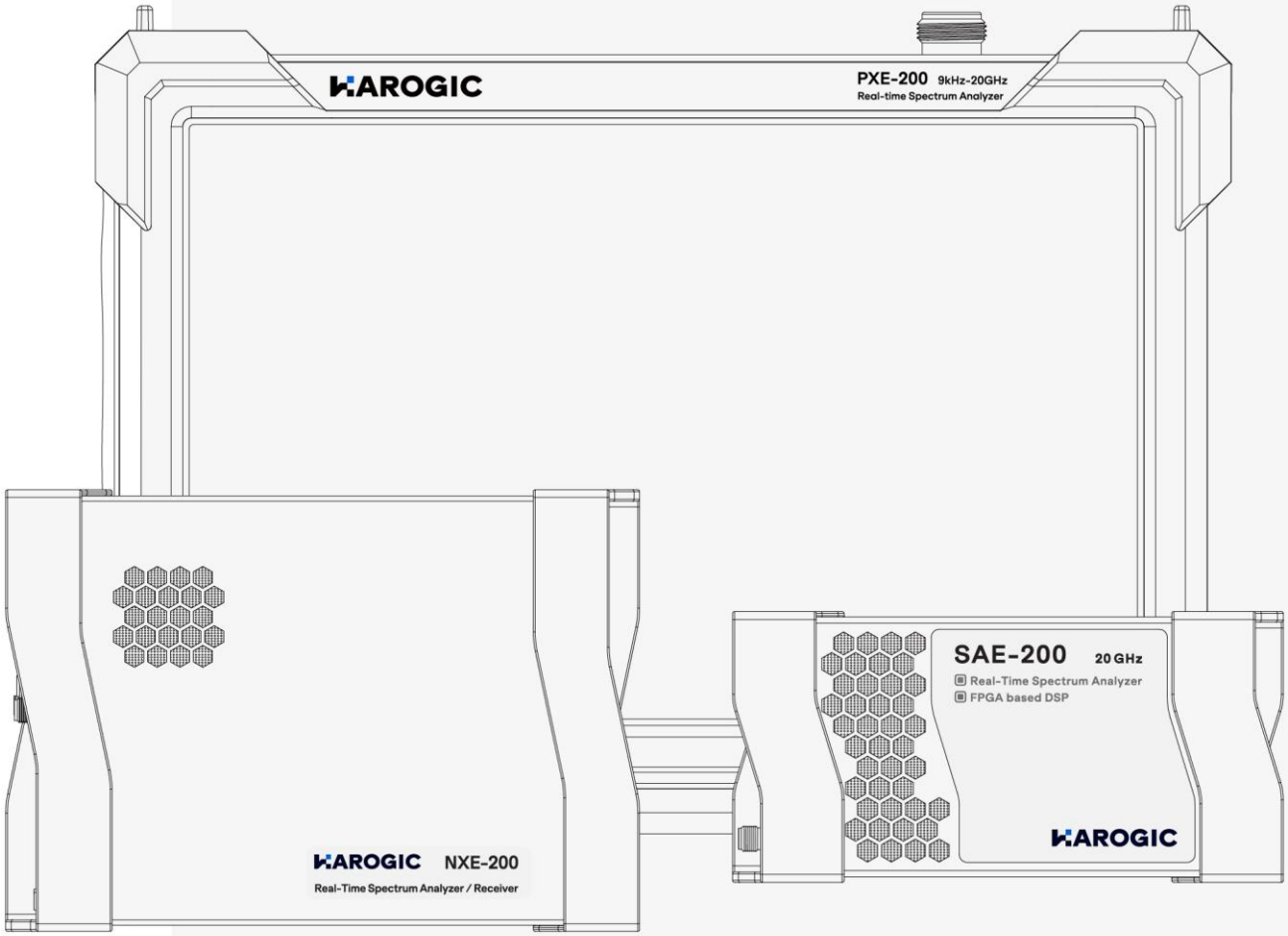
用户指南

HTRA API 编程指南



V0.55.82
2026/04/14

HAROGIC 海得逻捷

目录

1. 版本管理

版本更新说明表

版本号	内容	时间
V0.55.82	1. 增加：设备与系统 Device 章节中新增 APISupportFirmwareVersions、Device_ReEnumerate 函数	2026-04-13
V0.55.79	1. 增加：设备与系统 Device 章节中新增 Device_SetMcuSysPowerState、Device_QueryPowerSupplyState、Device_GetFreqPlanning、Device_SetIFOutput、Device_QueryVersion_PMU、Device_QueryVersion_AGU、Device_InitIFAGC 和 Device_GetAmpAttenState 函数 2. 重构：设备及 API 版本	2026-03-11
V0.55.77	1. 初始版本	2026-02-25

2. 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）、接收机/IQ流（IQS）、检波分析（DET）、实时频谱分析（RTA）、数字解调（选件）、列表扫描和相位噪声测量。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		
标准频谱分析	● 全景频谱扫描 ● 频谱监测 ● 相位噪声 ● 谐波测试	● 杂散测试 ● 信道功率测试 ● OBW、ACPR测试
接收机/IQ流	● 时域信号查看 ● IQ记录 ● AM解调	● FM解调 ● 用户应用
检波分析	● 脉冲信号观察	● 时间功率关系
实时频谱分析	● 突发信号观察 ● 隐秘信号发现	● 频谱动态观察
相位噪声测量	● 频率稳定性分析	● 近端噪声观察
数字解调	● 调制质量定量 ● 矢量特征分析	● ASK/FSK/GMSK/PSK/QAM解调
列表扫描	● 多频段频谱扫描 ● 分段参数设置	● 全景与细节观测 ● 多频段干扰排查

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，将设备配置至指定的工作模式;

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 API 的调用逻辑与调用地图章节。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。

3. 设备及 API 版本

系统是多个软固件的联合运行，在本系统中，涉及的软固件包括：设备主控固件（MFW/MCU）、FPGA固件（FFW）、总线固件（Bus）、PMU固件版本（PMU）、AGU固件版本（AGU）及应用程序接口（API）。

版本匹配原则

为确保系统稳定运行，上述各组件版本必须遵循严格的**基线版本对齐**原则。可参照下表进行版本的核对与部署。

表格 1 软固件基线版本对应表

API	FPGA	MCU	Bus	PMU	AGU
0.55.82	0.55.102	0.55.97	0.55.8	1.7	1.2
0.55.79	0.55.100	0.55.92	0.55.8	1.7	1.2
0.55.77	0.55.99	0.55.90	0.55.6	1.7	1.2

注意：1）为避免系统异常，切勿将不同基线版本中的软固件交叉使用，请务必严格按照上表所列版本进行统一部署或升级。

2）确保实际使用的API版本与手册封面所标识的对应，以免出现手册接口说明与实际功能不符的情况。

4. 函数类别简介

表格 2 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。
接收机/IQ 流 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。
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。
模拟解调 ADM	模拟解调类后处理函数，与硬件状态无关。 该类别对设备采集的 IQ 数据进行解调，包括 AM 解调、FM 解调等功能。
相位噪声测量 PNM	该模式下，接收机对获取到的数据进行相位噪声测量分析，实时输出载波特性，相噪迹线等结果。
数字解调 Demod	IQS 模式数据获取后处理函数，与硬件状态无关。 该类别对设备采集的 IQ 数据进行解调，返回星座图、眼图和 EVM 等。
脉冲检测 Pulse	IQS/DET 模式数据获取后处理函数，与硬件状态无关。 该类别对采集的时域数据进行脉冲检测，可输出脉宽、周期、占空比等参数。
列表扫描 MSCAN	该模式下，频谱分析仪只需一次配置，即可实现多频段的顺序扫描，并同步输出频谱与 IQ 数据。系统支持对各频段的参数进行完全独立配置（如参考电平、抽取倍数、FFT 点数、IFAGC 等）。适用于多频段监控，干扰检测等应用场景。

5. API 的调用逻辑与调用地图

5.1 标准频谱分析（SWP）的 API 调用地图

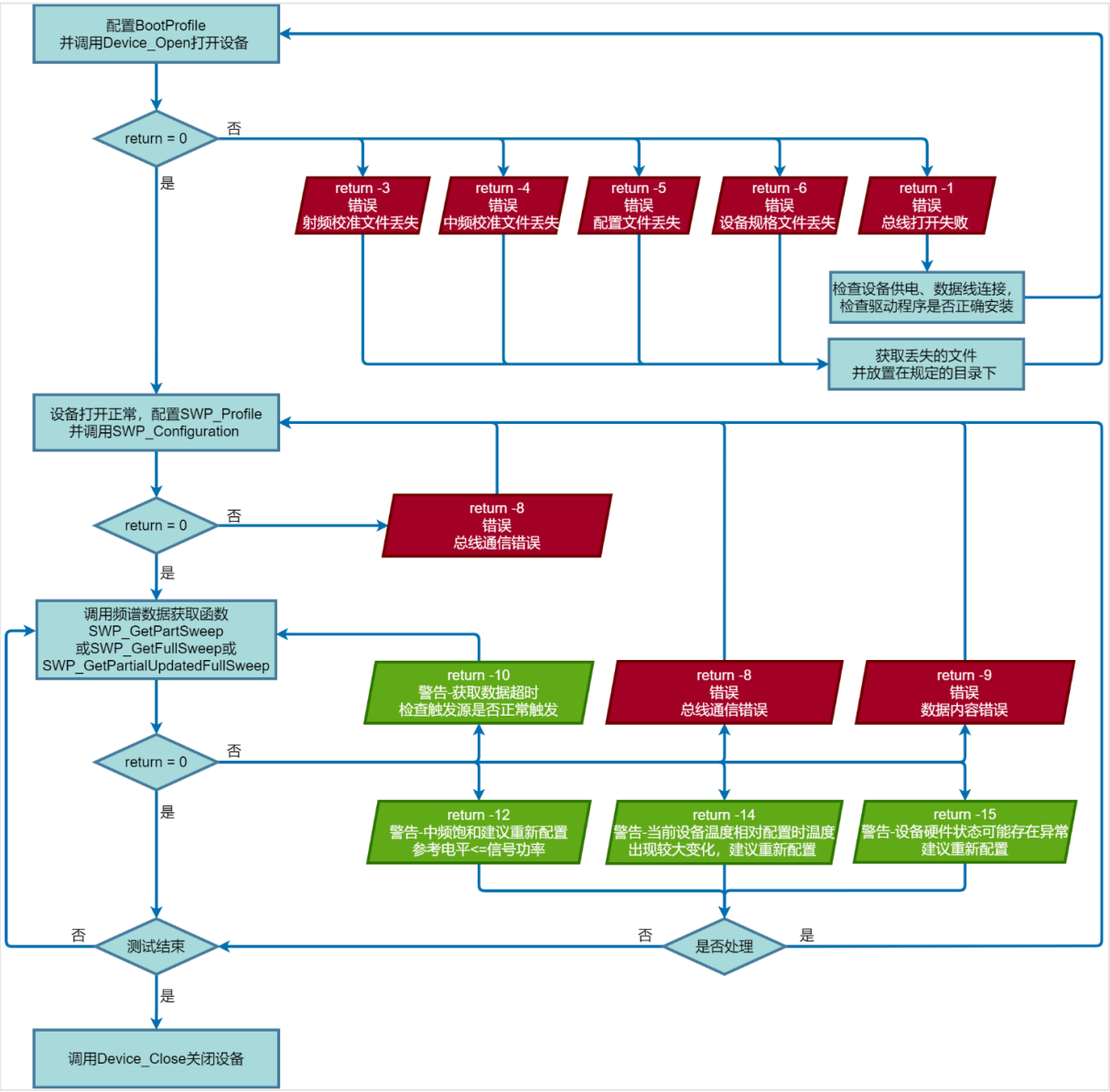


图 2 SWP 模式调用流程图

5.2 接收机/IQ 流（IQS）的 API 调用地图

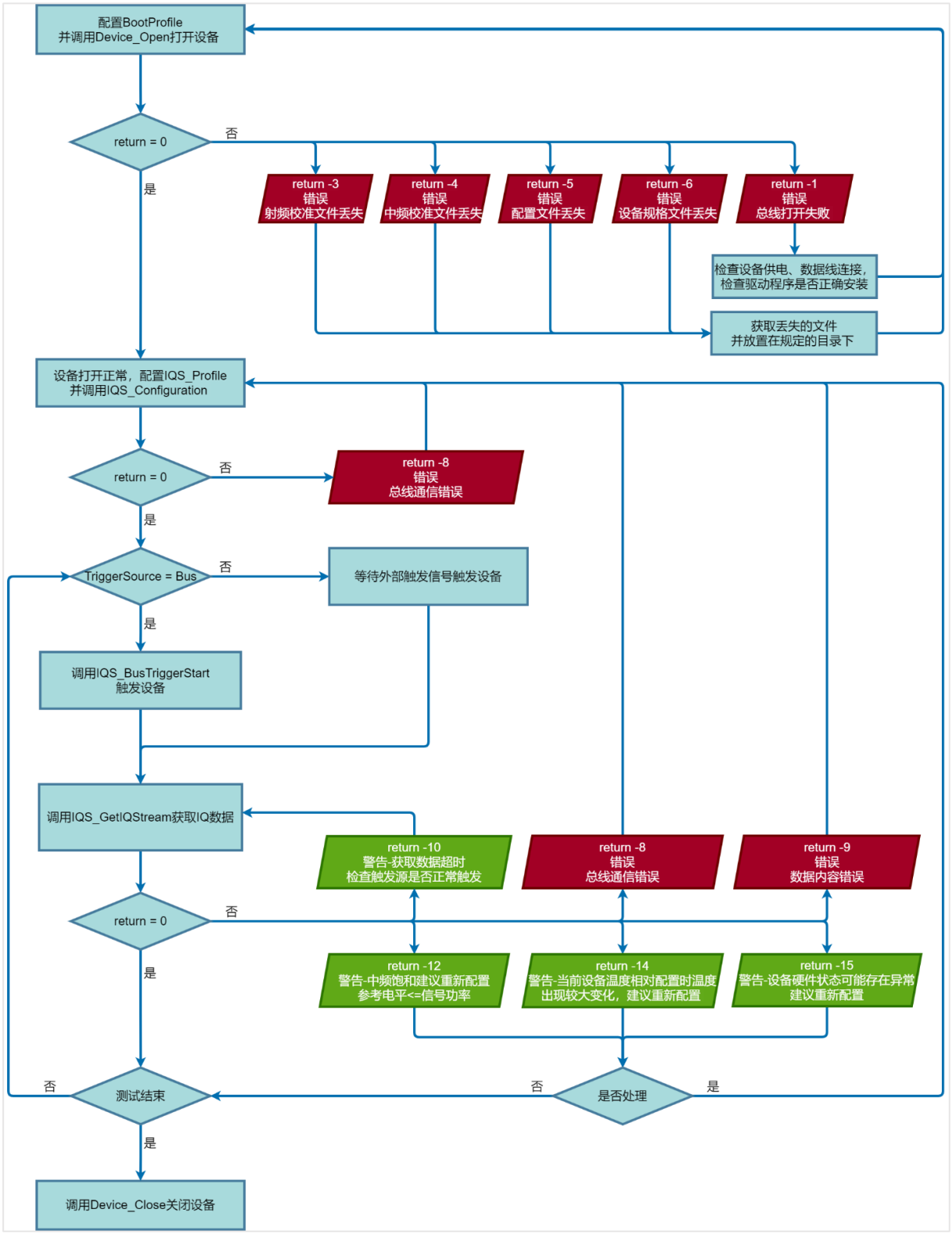


图 3 IQS 模式调用流程图（触发模式为 Fixed）

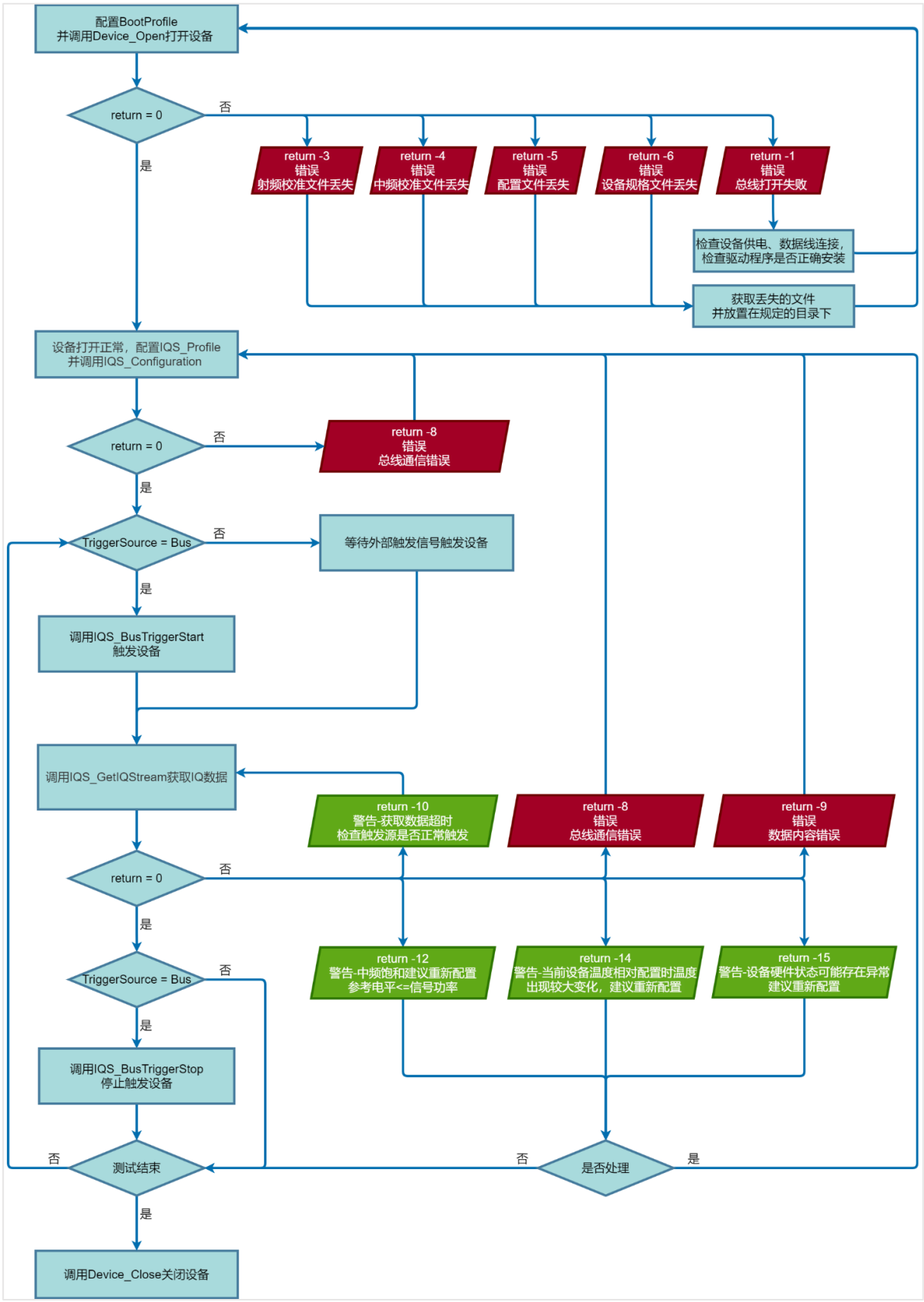


图 4 IQS 模式调用流程图（触发模式为 Adaptive）

5.3 检波分析（DET）的 API 调用地图

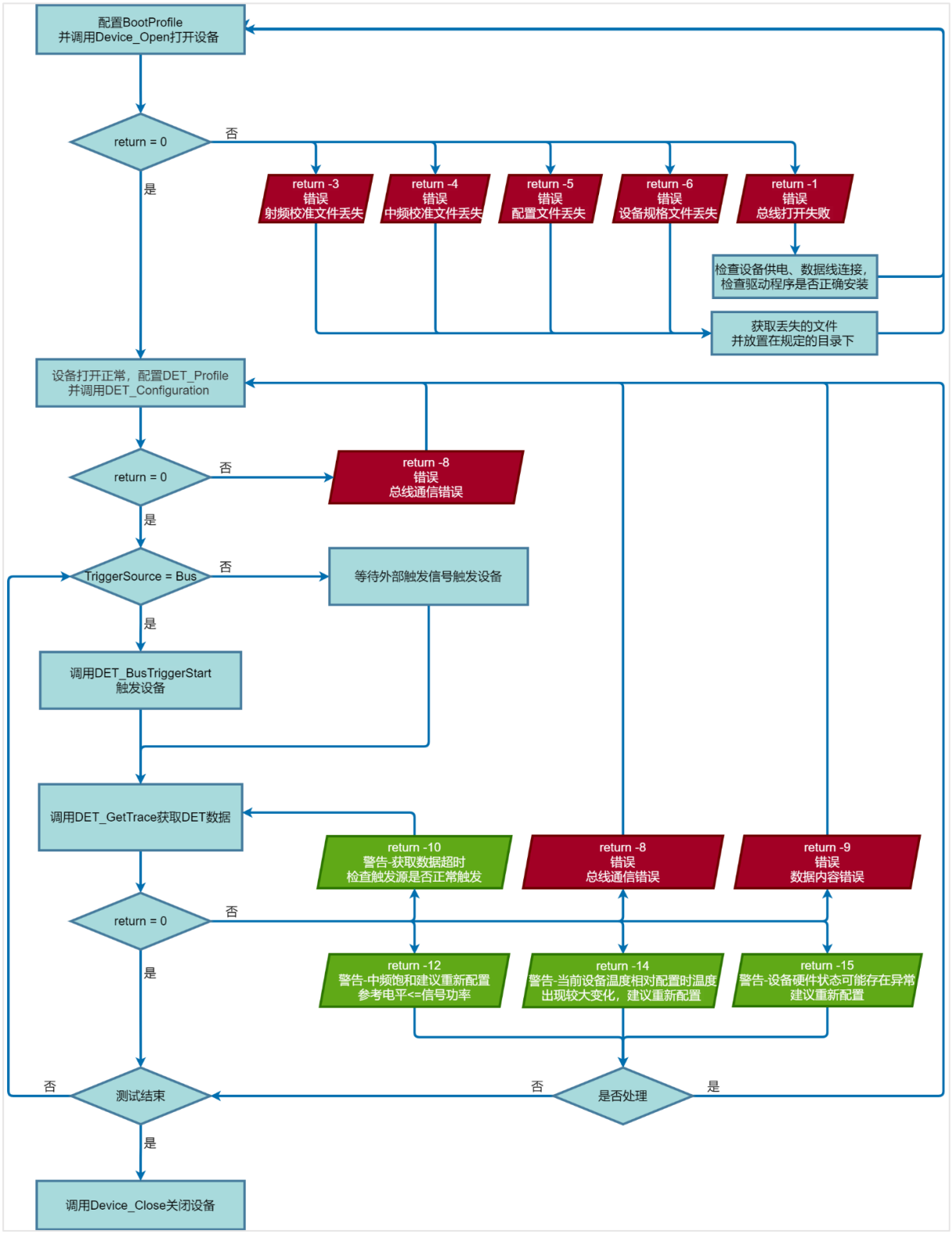


图 5 DET 模式调用流程图（触发模式为 Fixed）

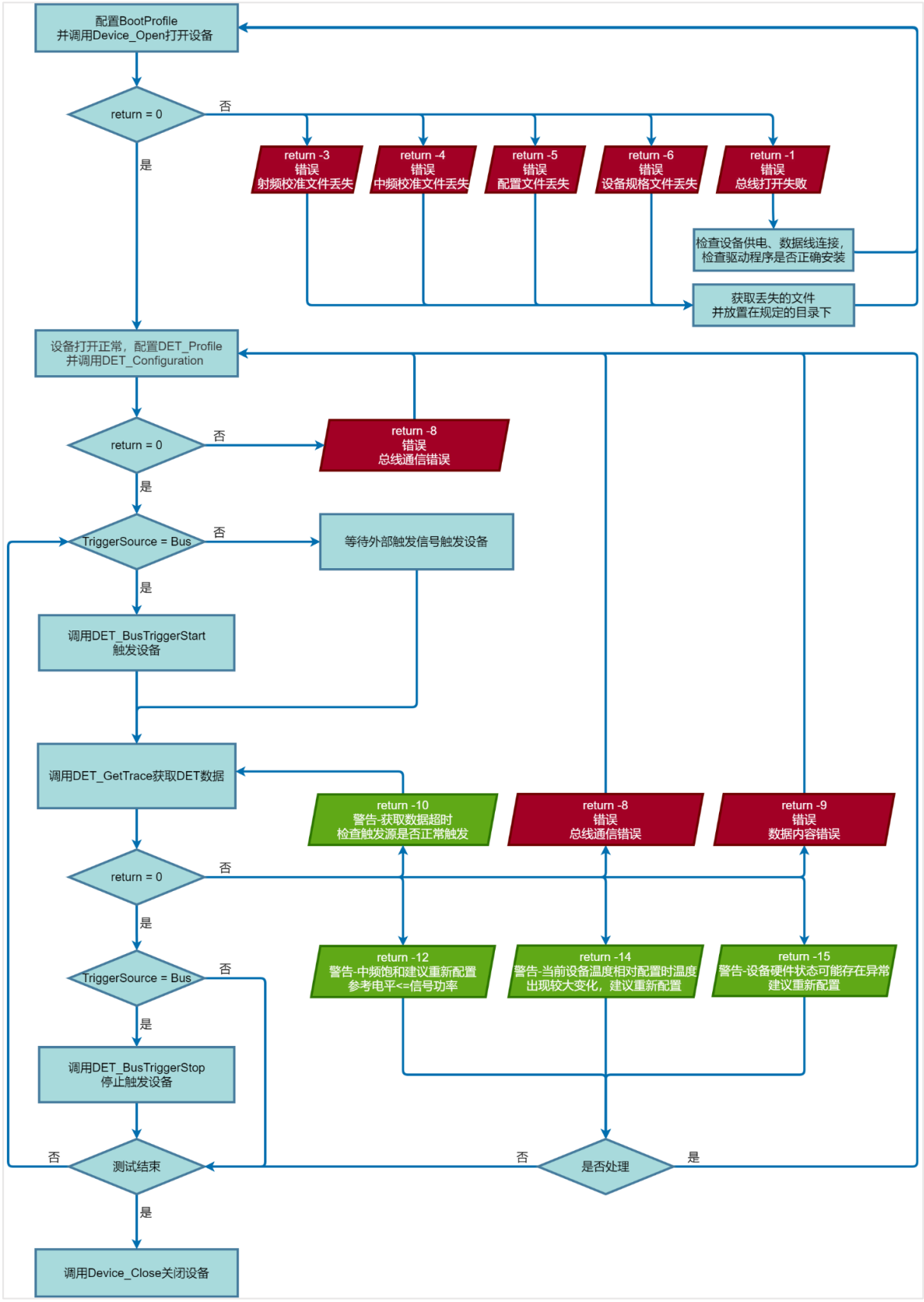


图 6 DET 模式调用流程图（触发模式为 Adaptive）

5.4 实时频谱分析（RTA）的 API 调用地图

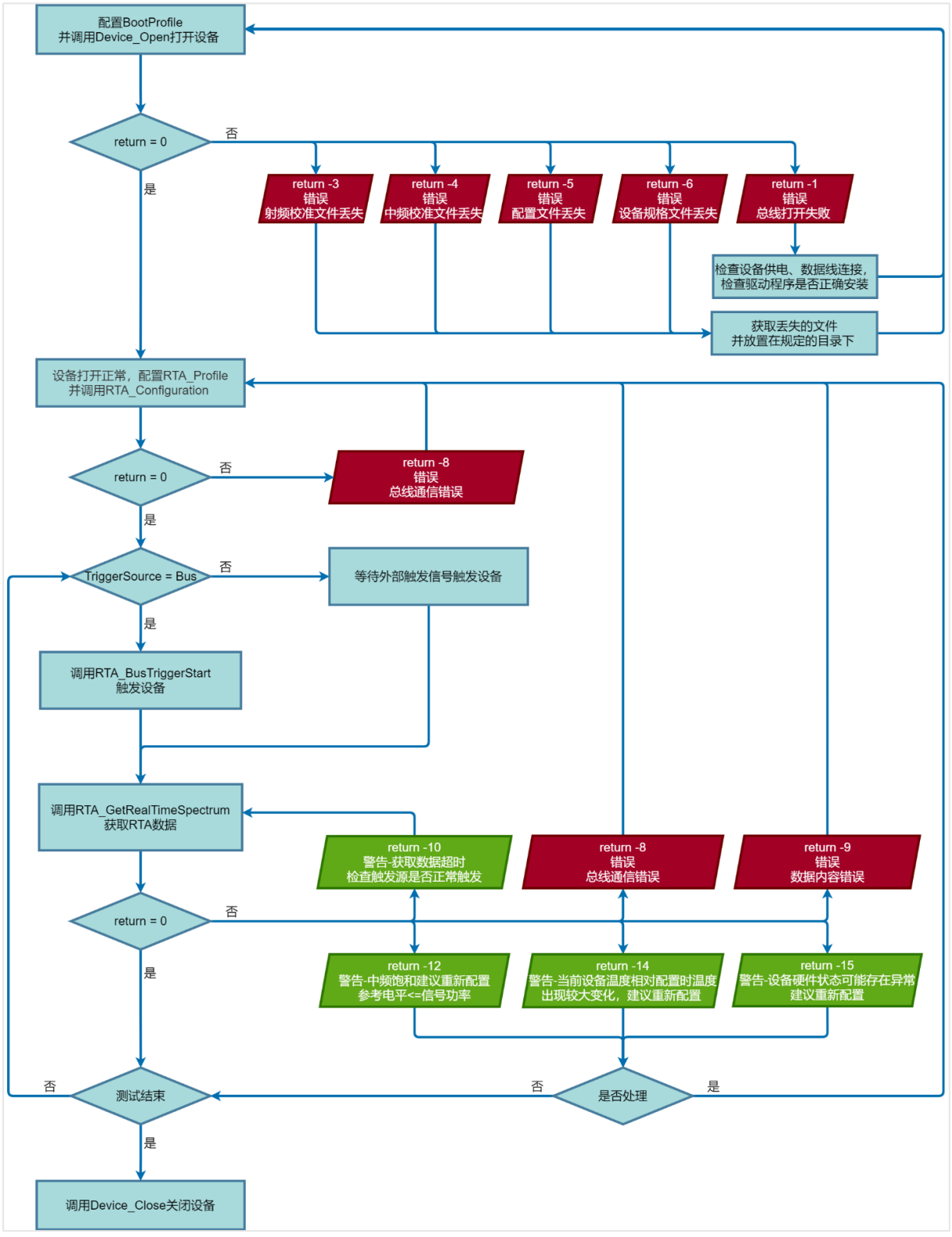


图 7 RTA 模式调用流程图（触发模式为 Fixed）

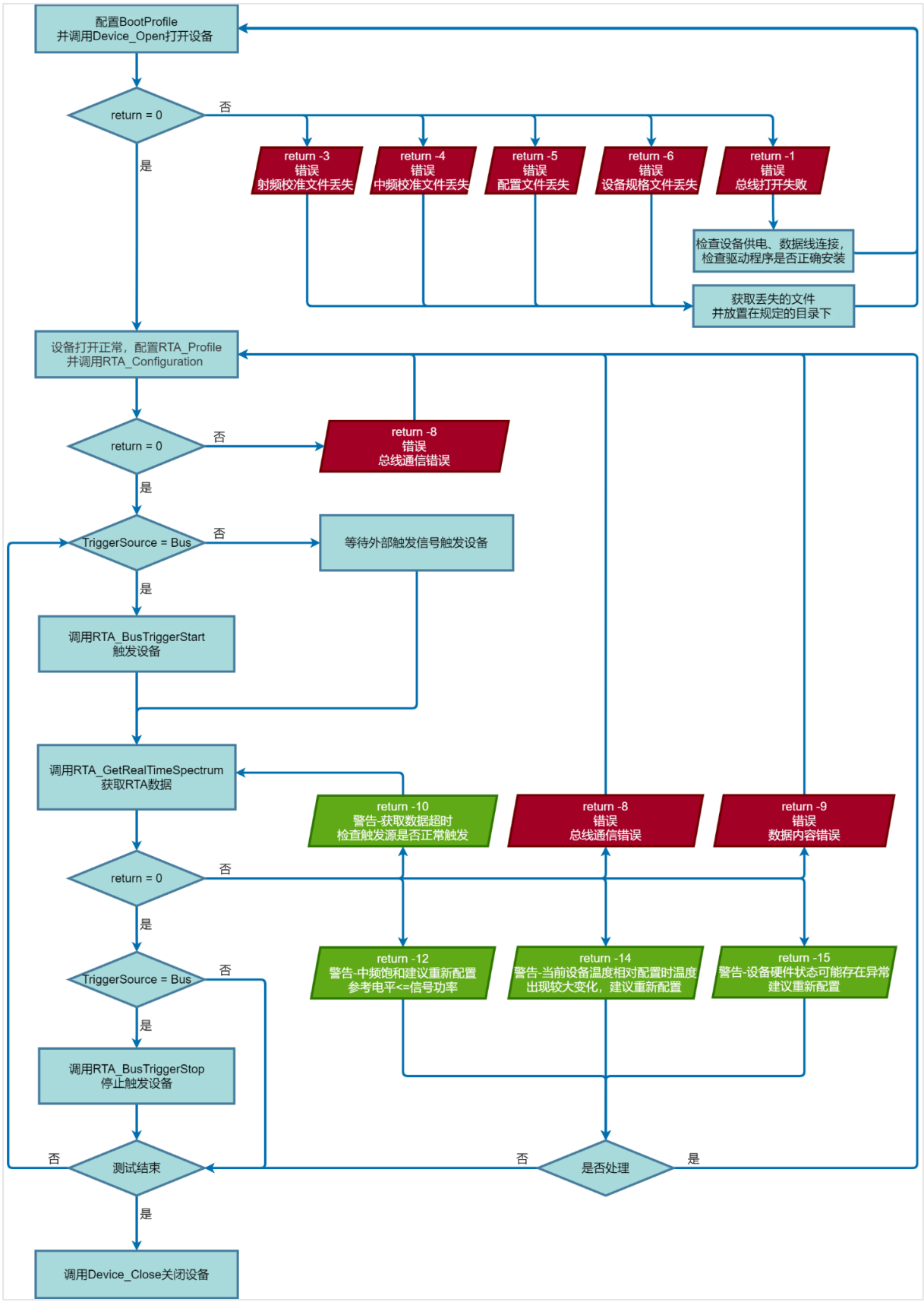


图 8 RTA 模式调用流程图（触发模式为 Adaptive）

6. 重要变量/重要设置及其概念

本章罗列了频谱分析仪/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。

6.1 系统

表格 3 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。

6.2 幅度

表格 4 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	系统默认根据参考电平来自动配置衰减器和前置放大器。参考电平可以简单理解为系统不饱和所能接受的最大输入功率。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-3dBm 的单音信号，则设置参考电平为 0dBm，可获得良好的观察动态。
2	衰减 Atten	SWP/IQS/ DET/RTA	衰减默认为自动（Atten = -1），此时系统仅由参考电平指定通道衰减。当需要手动设置通道衰减时，请设置 Atten 为指定值。
3	前置放大器 Preamplifier	SWP/IQS/ DET/RTA	对于带有前置放大器的设备，是否启用前置放大器会明显影响系统的噪声性能和线性度。 启用前置放大器：可降低系统噪声，但会减小系统可承受的最大线性输入功率以及最大损毁功率。 关闭前置放大器：可提高可承受的输入功率上限，但系统噪声会相对较高。 系统通常可以配置为： 根据参考电平自动控制前置放大器的启用或关闭； 强制关闭前置放大器，以避免因过载而造成损坏。

4	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。不同的模拟中频带宽具有不同的带外抑制、带内平坦度、群延时等性能，请根据应用需要选择合适的中频档位。
5	中频增益档位 IFGainGrade	SWP/IQS/ DET/RTA	系统允许用户调节中频增益，以在杂散、线性度和噪声水平之间进行优化。档位序号越高，中频增益越大，相邻档位通常相差1 dB ~ 3 dB。 系统总增益 = 射频增益 + 中频增益。在参考电平不变（总增益固定）的情况下: 提高中频增益 → 混频器输入功率降低 → 杂散性能改善、线性度提升，但噪声性能会变差。 降低中频增益 → 混频器输入功率升高 → 杂散性能和线性度可能变差，但噪声性能会改善。 当射频增益已达最大（即参考电平很低，如-60 dBm）: 进一步提高中频增益会提升系统总增益，从而可能改善噪声性能。 当射频增益未达最大（如参考电平0 dBm）: 提高中频增益 → 优化杂散和线性度，但噪声性能恶化; 降低中频增益 → 杂散和线性度变差，但噪声性能改善。

6.3 频率

表格 5 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	在 SWP 模式下，系统的频谱分析方式由TracePointStrategy决定: 当TracePointStrategy = BinSizeAssined时：系统采用扫描法进行频谱分析，迹线点数由TracePoints指定，此时TraceAlign设置无效。 TracePointStrategy为其他值时：系统采用FFT分析法进行频谱分析。由于底层软件实现和迹线检波机制的限制，
4	迹线点数 TracePoints		
5	迹线对齐 TraceAlign		

		原生分析频率点无法精确对齐到配置的起始与终止频率。返回的迹线数据点数会 略微超出 配置的频率范围。 对此，用户可按需处理: 使用DSP_InterceptSpectrum函数对返回频谱数据进行截取，以匹配所需频段；设置TraceAlign = AlignToStart，将迹线的实际起始频率与配置的起始频率对齐（但终止频率仍需截取处理）。 在 FFT 分析法下，用户可以设定期望的迹线点数（TracePoints），但由于底层实现的限制，实际返回的迹线点数通常无法精确达到设定值，系统会返回一个最接近可用的点数。

6.4 分析

表格 6 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	系统提供关闭、标准和增强三种杂散抑制模式。该功能可有效抑制大部分组合分量杂散（对系统的剩余响应无优化效果），但会降低扫描速度和时变信号的测量能力。 对于稳态信号（如单音信号）测试，开启杂散抑制能显著提升无杂散动态范围。 对于快速时变信号（如调制信号）测试，开启该功能可能导致信号偶发丢失或功率测量不准确，因此需谨慎使用。建议通过切换功能开关并观察频谱变化，判断当前测试场景是否适合开启杂散抑制。
2	功耗平衡 PowerBalance	SWP	在 SWP 模式下，用户可以通过设置功耗平衡参数在扫描速度与设备功耗之间进行权衡: 功耗平衡 = 0：系统以 最高扫描速度 运行; 功耗平衡 = 40~1000（常规设置值）：数值越大，扫描速度越低，同时 功耗也越低。 需要特别注意，功耗平衡数值越高，时变信号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，系统提供多种窗函数，不同窗型各有优势，请根据测试需求选择: FlatTop 窗: 具备优良的幅度准确性，可减小栅栏效应带来的幅度误差，适用于对幅度精度要求较高的测试场合。

			Blackman-Nuttall 窗：主瓣窄，频率分辨率高，在相同 RBW 下扫描速度快于 FlatTop 窗，适用于高频率分辨率和快速扫描的测试场景。 LowSideLobe 窗：具有极低的旁瓣电平，能有效抑制强信号对邻近频率的干扰，适用于动态范围要求高、强弱信号共存的测试场景。
4	FFT 执行策略 FFTExcutionStrategy	SWP	在标准频谱分析模式下，用户可以选择以下信号处理的运算方式： 自动模式：由系统根据 RBW 自动选择 FPGA 或 CPU 运算； 仅 FPGA 运算：显著降低对 CPU 的处理负载，但由于单次 FFT 点数受限，在 RBW≤5kHz 时扫描速度较慢； 仅 CPU 运算：允许使用超过 64k 点的 FFT，在中小 RBW（≤5kHz）时可获得更高的扫描速度。
5	扫描时间 SweepTime	SWP/RTA	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 = SWTMode_Manual 时，该参数为绝对时间；当指定为*N 时，该参数为扫描时间倍率，即按照最小扫描时间的 N 倍进行扫描。
6	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。
7	总线超时 BusTimeOut	IQS/DET/RTA	总线超时用于为获取数据的相关函数设定一个执行时间上限。如果系统在该时间内未能获取到有效数据，就会强制返回，防止系统无限期等待。

6.5 检波器和迹线检波器

检波器：在同一个本振频点下，采集检波比帧数据，按照检波器特性，对多帧数据逐频点检波，最终生成特征值帧。下图以PosPeak Detector为例，介绍正峰值检波的过程，其中Frame 0、Frame 1、Frame 2为不同时刻采集的数据，After PosPeak Detector为正峰值检波后的数据。

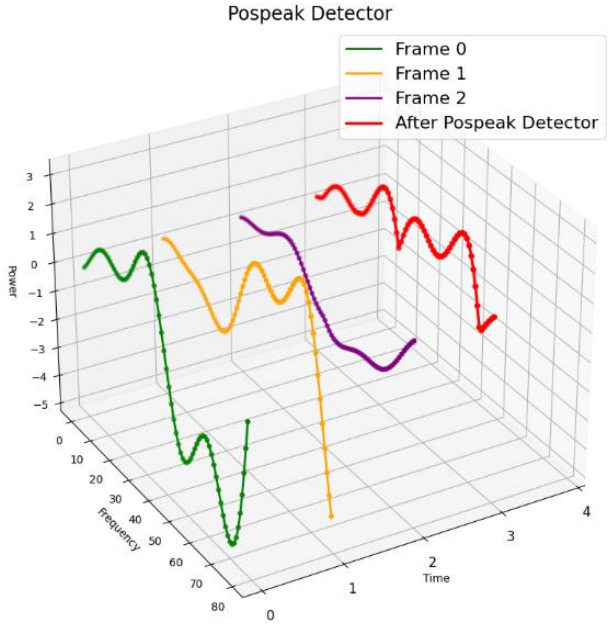


图 9 正峰值检波

迹线检波器：根据选定的迹线检波器，以迹线检波比为步进，对整条频谱迹线进行检波，从而生成特征值迹线。下图以PosPeak TraceDetector为例，介绍正峰值迹线检波的过程，其中Before PosPeak Trace Detector为正峰值迹线检波前的数据，After PosPeak Trace Detector为正峰值迹线检波后的数据。

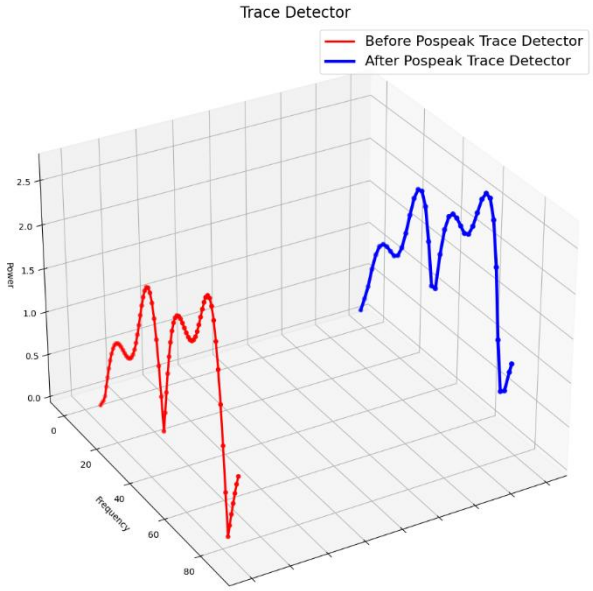


图 10 正峰值迹线检波

6.6 默认单位

表格 7 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

7. 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。

7.1 Device_List

int Device_List(const BootProfile_TypeDef* BootProfile, uint8_t* Devicecount, uint8_t DevNum[MAX_DEVICE], DeviceInfo_TypeDef DeviceInfo_O[MAX_DEVICE])	
功能描述	
查询当前连接在上位机上的所有 USB 型设备对应的设备号和设备信息。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] BootProfile	设备启动配置，指定接口类型为 USB 型。 详见 BootProfile_TypeDef 结构体的定义。
[out] Devicecount	返回当前挂载在上位机上设备的数量，总线挂载的设备数量上限为 256 台。
[out] DevNum[MAX_DEVICE]	返回当前挂载在上位机所有设备的编号。
[out] DeviceInfo_O[MAX_DEVICE]	返回当前所有挂载设备的信息，有效信息包括 model 和 uid（低 64bit）。 详见 DeviceInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	所有设备的总线版本（Bus）需达到 0.55.8 及以上
示例	

```
int Status = 0; int DevNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
uint8_t Devicecount = 0;
DeviceInfo_TypeDef DeviceInfo_O[MAX_DEVICE];
vector<uint8_t>DevNumList(MAX_DEVICE);
```

```
Status = Device_List(&BootProfile, &Devicecount, DevNumList.data(), DeviceInfo_O);
for (int i = 0; i < Devicecount; i++) {
    if (DeviceInfo_O[i].Model == 57) { // 57 为设备型号，根据实际情况修改
        DevNum = DevNumList[i];
        break;
    }
}
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
```

7.2 Get_APIVersion

int Get_APIVersion(void)	
功能描述	
获取当前使用的 API 库的版本号。	
兼容性	0.55.77 及之后版本支持
返回值	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]表示修订版本。
调用约束	无。
示例	

```
int apiVersion = Get_APIVersion();
int major = 0, minor = 0, rev = 0;
major = (apiVersion >> 16) & 0xffff;
minor = (apiVersion >> 8) & 0xff;
rev = apiVersion & 0xff;
cout << "htra_api verion: " << major << "." << minor << "." << rev << endl;
```

7.3 APISupportFirmwareVersions

int APISupportFirmwareVersions (DeviceFirmwareVersion_TypeDef** Versions uint32_t* Count)	
功能描述	
获取当前 API 支持的固件版本。	
兼容性	0.55.79 及之后版本支持
参数说明	

[out] Versions	返回当前 API 支持的固件版本信息结构体的首地址 详见 DeviceFirmwareVersion_TypeDef 结构体的定义。
[out] Count	输出当前 API 所支持的固件版本信息组数
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	无
示例	

```
DeviceFirmwareVersion_TypeDef* pVersions = nullptr;
uint32_t count = 0;
int status = APISupportFirmwareVersions(&pVersions, &count);
if (status == 0) {
  for (uint32_t i = 0; i < count; ++i) {
    cout << "FPGA: " << pVersions[i].FFWVersion << " | MCU: " <<
    pVersions[i].MFWVersion << " | Bus: " << pVersions[i].BusVersion << " | PMU: " <<
    pVersions[i].PMUVersion << " | AGU: " << pVersions[i].AGUVersion << std::endl;
  }
}
```

7.4 Device_ReEnumerate

int Device_ReEnumerate(void** Device)	
功能描述	
当设备接至 USB 3.0 接口上电瞬间被识别为 USB 2.0，但已确认通过重新插拔可以正确识别为 USB 3.0。在此前提下，调用本函数可直接在软件层面完成从 USB 2.0 到 USB 3.0 的重新枚举，无需再进行物理上的重新插拔。 本函数成功后，需要调用 Device_Close 函数，然后再调用 Device_Open，以恢复和设备的连接。	
兼容性	0.55.82 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

```
while (1)
{
    int Status = 0; void* Device = NULL; int DevNum = 0;
    BootProfile_TypeDef BootProfile;
```

```
    BootInfo_TypeDef BootInfo;
    BootProfile.DevicePowerSupply = USBPortAndPowerPort;
    BootProfile.PhysicalInterface = USB;
    Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
    // 判断 USB 接口是否被识别为 3.0
    if (BootInfo.BusSpeed == 2) {
        Device_ReEnumerate(&Device);
        Device_Close(&Device);
        std::this_thread::sleep_for(std::chrono::seconds(20));
    }
    else {
        break;
    }
}
```

7.5 Device_Open

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

功能描述

打开指定设备并获取设备句柄，用于后续 API 调用中标识目标设备。当系统中存在多台设备时，可通过不同的设备号分别打开对应设备。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] DeviceNum	指定设备编号。多台设备时可用该编号选择目标设备，编号从 0 开始递增。
[in] BootProfile	设备启动配置，包含接口类型、供电方式及网络参数，用于初始化设备。 详见 BootProfile_TypeDef 结构体的定义。
[out] BootInfo	返回设备启动信息，包括设备信息、总线速度与版本、API 版本，以及启动过程中的错误和警告。 详见 BootInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。

调用约束	需要在对设备执行其他函数调用之前调用一次 Device_Open，以获取设备资源和设备句柄。 同一模块中只需调用一次，后续操作可使用返回的设备句柄执行相关操作。 在模块使用结束后，必须调用 Device_Close 释放设备资源。
示例	请参考 Device_QueryDeviceInfo_Realtime()函数的相关示例。

7.6 Device_Close

int Device_Close(void** Device)	
功能描述	
关闭指定设备并释放由 Device_Open 分配的资源，在设备操作完成后使用。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	仅需在程序执行结束时调用此函数。调用后 USB 设备连接关闭，内存空间释放。如需重新使用设备，必须再次调用 Device_Open 建立连接并打开设备。
示例	请参考 Device_QueryDeviceInfo_Realtime()函数的相关示例。

7.7 Device_QueryDeviceState/Device_QueryDeviceState_Realtime

int Device_QueryDeviceState(void** Device, DeviceState_TypeDef* DeviceState)
int Device_QueryDeviceState_Realtime(void** Device, DeviceState_TypeDef* DeviceState)
功能描述
获取频谱仪设备状态，包括设备温度、硬件工作状态及地理时间信息（需选件支持）。 Device_QueryDeviceState：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； Device_QueryDeviceState_Realtime：实时获取，短时间内会占用数据通道。

兼容性	0.55.77 及之后版本支持
参数说明	

[in] Device	设备句柄。
[out] DeviceState	指向设备状态信息的结构体指针。调用函数后，该结构体会被更新为最新的设备状态信息。 详见 DeviceState_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在调用 Device_Open 后使用此函数。
示例	请参考 Device_QueryDeviceInfo_Realtime()函数的相关示例。

7.8 Device_QueryDeviceInfo/Device_QueryDeviceInfo_Realtime

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

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

功能描述

获取设备信息，包括设备序列号、硬件版本和固件版本等相关信息。

Device_QueryDeviceInfo：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。

Device_QueryDeviceInfo_Realtime：实时获取，短时间内会占用数据通道。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] DeviceInfo	设备基本信息结构体，包含设备唯一标识和各类版本信息。调用函数后，该结构体会被更新为最新的设备信息。 详见 DeviceInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

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

```
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
DeviceState_TypeDef DeviceState;
Status = Device_QueryDeviceState(&Device, &DeviceState);
Status = Device_QueryDeviceState_Realtime(&Device, &DeviceState);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo(&Device, &DeviceInfo);
Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceInfo);
Status = Device_Close(&Device);
```

7.9 Device_QueryPowerSupplyState

```
int Device_QueryPowerSupplyState(
    void** Device,
    PowerSupplyState_TypeDef* PowerSupplyState
)
```

功能描述

获取设备的供电状态，返回射频板电源端口（POWER）和数字板 USB 端口（DATA）的电压与电流。

兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] PowerSupplyState	返回设备电源与数据口的电压和电流。 详见 PowerSupplyState_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在调用 Device_Open 后使用此函数。

示例

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

8. 设备与系统 Device 其他函数

8.1 Device_SetSysPowerState

int Device_SetSysPowerState(void** Device, SysPowerState_TypeDef SysPowerState)	
功能描述	
设置设备的系统电源状态，可控制系统各工作区的上电、射频模块下电或射频模块待机，以满足不同功耗和唤醒需求。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] SysPowerMode	设置设备系统电源状态。 详见 SysPowerState_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	
int Status = -1; int DeviceNum = 0; void* Device = NULL; BootProfile_TypeDef BootProfile; BootProfile.DevicePowerSupply = USBPortAndPowerPort; BootProfile.PhysicalInterface = USB; BootInfo_TypeDef BootInfo; Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo); SysPowerState_TypeDef SysPowerState = PowerON; Status = Device_SetSysPowerState(&Device, SysPowerState); Status = Device_Close(&Device);	

8.2 Device_SetMcuSysPowerState

int Device_SetMcuSysPowerState (void** Device, SysPowerMode_TypeDef SysPowerState)
功能描述
设置设备的 MCU 低功耗电源状态，可控制 MCU 的上电、待机和掉电模式，以满足不同功耗需求。

兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] SysPowerState	设置设备 MCU 低功耗电源状态。 详见 SysPowerMode_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

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

8.3 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

通过已知周期的外部触发信号校准设备参考时钟频率。函数可根据指定触发周期和触发次数计算校准后的频率，并可选择将校准结果重写入校准文件以作为后续启动的默认参考频率。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] ClkCalibrationSource	指定时钟校准源。

	详见 ClkCalibrationSource_TypeDef 枚举的定义。
[in] TriggerPeriod_s	已知校准信号的周期（单位：秒）。该值精度直接影响参考时钟的校准精度。
[in] TriggerCount	用于校准的触发次数。对于使用 GNSS 1PPS 校准的场景，触发次数越多可更好抑制抖动误差，提高校准精度，但校准时间也越长。建议使用 GNSS 1PPS 时触发次数大于 30（即校准时间超过 30 秒）。
[in] RewriteRFCal	是否将校准结果写入校准文件。 0：不写入，设备下电后校准结果失效。 1：写入，设备上下电后仍保持校准状态。
[out] RefCLKFreq_Hz	返回本次校准得到的参考时钟频率（单位：Hz）。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
ClkCalibrationSource_TypeDef ClkCalibrationSource = CalibrateByExternal;
double TriggerPeriod_s = 1; uint64_t CalibrationTimes = 1 * 60;
bool RewriteRFCal = false; double RefCLKFreq_Hz = 0;
Status = Device_CalibrateRefClock(&Device, ClkCalibrationSource, TriggerPeriod_s, CalibrationTimes, RewriteRFCal, &RefCLKFreq_Hz);
Status = Device_Close(&Device);	

8.4 Device_SetFanState

```
int Device_SetFanState(
    void** Device,
    const FanState_TypeDef FanState,
    const float ThreshouldTemperature
)
```

功能描述
设置设备风扇的工作模式，并根据指定温度阈值控制风扇启停，以实现设备温控管理。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] FanState	设置设备风扇工作模式。 详见 FanState_TypeDef 枚举的定义。
[in] ThreshouldTemperature	门限温度（摄氏度），当 FanState = 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 10℃时，关闭风扇。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
FanState_TypeDef FanState = FanState_On;
float Temperature = 0;
Status = Device_SetFanState(&Device, FanState, Temperature);
Status = Device_Close(&Device);
```

8.5 Devcie_SetFreqResponseCompensation

```
int Devcie_SetFreqResponseCompensation(
    void** Device,
    uint8_t State,
    const double *Frequency_Hz,
    const float *CorrectVal_dB,
    uint8_t Points
)
```

功能描述

在 SWP 模式下，采用频率补偿机制对指定频段进行功率修正。补偿规则如下：

1. 频率补偿区间内插值补偿：

在 Frequency_Hz 数组中，相邻频率点之间采用线性插值进行补偿，确保功率补偿值在指定频率范围内平滑过渡。

2. 起始频率至补偿数组起始频率(Frequency_Hz[0])：

该区间的补偿值为 CorrectVal_dB[0]，即使用补偿数组的起始补偿值进行填充。

3. 补偿数组终止频率(Frequency_Hz[Points - 1]）至终止频率： 该区间的补偿值为 CorrectVal_dB[Points - 1]，即使用补偿数组的终止补偿值进行填充。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] **Device**	设备句柄。
[in] **State**	是否开启频率响应补偿。0：关闭补偿；1：开启补偿。
[in] **Frequency_Hz**	频率补偿数组。
[in] **CorrectVal_dB**	功率补偿数据数组。
[in] **Points**	补偿点数，上限 256 个点。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Configuration 后调用此函数。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.StartFreq_Hz = 2e8;
SWP_ProfileIn.StopFreq_Hz = 3e8;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
uint8_t compensation_points = 4;
vector<double> Compensate_freq = { 239.999e6, 240e6, 260e6, 260.001e6 };
vector<float> Compensate_dBm = { 0.0f, 10.0f, 30.0f, 0.0f };
Status = Devcie_SetFreqResponseCompensation(&Device, 1, Compensate_freq.data(), Compensate_dBm.data(), 4);
MeasAuxInfo_TypeDef MeasAuxInfo;
while (1) {
```

```
    Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &
MeasAuxInfo);
}
Status = Device_Close(&Device);
```

8.6 Device_GetNetworkDeviceList

```
int Device_GetNetworkDeviceList(
    uint8_t*DeviceCount,
    NetworkDeviceInfo_TypeDef DeviceInfo[64],
    uint8_t LocalIP[4],
    uint8_t LocalMask[4]
)
```

功能描述

在使用网口型设备时，获取网络中所有设备的 IP 地址和子网掩码信息，同时返回本地网络接口的 IP 地址和子网掩码。

兼容性	0.55.77 及之后版本支持
参数说明	
[out] DeviceCount	返回网络中检测到的设备数量。
[out] DeviceInfo[64]	返回网络中各设备的信息，包括设备序列号、设备类型、硬件版本、MCU 和 FPGA 固件版本，以及 IP 地址和子网掩码。 详见 NetworkDeviceInfo_TypeDef 结构体的定义。
[out] LocalIP[4]	输出本地 IP 地址。
[out] LocalMask[4]	输出本地子网掩码。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	无。

示例

```
int Status = -1; uint8_t DeviceCount = 0; uint8_t LocalIP[4]; uint8_t LocalMask[4];
NetworkDeviceInfo_TypeDef DeviceInfo[64];
Status = Device_GetNetworkDeviceList(&DeviceCount, DeviceInfo, LocalIP, LocalMask);
```

8.7 Device_SetNetworkDeviceIP

int Device_SetNetworkDeviceIP(const uint64_t DeviceUID, const uint8_t IPAddress[4], const uint8_t SubnetMask[4])	
功能描述	
使用网口型设备时，通过设备的唯一序列号配置网络设备的 IP 地址和子网掩码，用于网络参数设置和设备通信管理。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DeviceUID	指定设备的序列号。
[in] IPAddress[4]	输入要配置的 IP 地址。
[in] SubnetMask[4]	输入要配置的子网掩码。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	
int Status = -1; uint64_t DeviceUID = 31325119004c0048; uint8_t IPAddress[4] = { 192, 168, 2, 100}; uint8_t SubnetMask[4] = {255, 255, 255, 0}; Status = Device_SetNetworkDeviceIP(DeviceUID, IPAddress, SubnetMask);	

8.8 Device_SetNetworkDeviceIP_PM1

int Device_SetNetworkDeviceIP_PM1(const uint8_t DeviceIP[4], const uint8_t IPAddress[4], const uint8_t SubnetMask[4])	
功能描述	
使用网口型设备时，通过指定设备的当前 IP 地址配置其新的 IP 地址和子网掩码，用于网络参数更新和设备通信管理。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DeviceIP[4]	输入设备当前的 IP 地址，用于定位设备。

[in] IPAddress[4]	输入要配置的新 IP 地址。
[in] SubnetMask[4]	输入要配置的新子网掩码。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	无。
示例	
int Status = -1; uint8_t DeviceIP[4] = {192, 168, 1, 100}; uint8_t IPAddress[4] = { 192, 168, 2, 100}; uint8_t SubnetMask[4] = {255, 255, 255, 0}; Status = Device_SetNetworkDeviceIP_PM1(DeviceIP, IPAddress, SubnetMask);	

8.9 Device_GetFullUID

int Device_GetFullUID(void** Device, uint64_t* UID_L64, uint32_t* UID_H32)
功能描述
获取设备完整的 UID 信息，包括高 32 位和低 64 位。以非实时方式获取，不会中断数据采集，信息在获取数据包后更新。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UID_L64	输出设备 UID 的低 64 位。
[out] UID_H32	输出设备 UID 的高 32 位。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	
int Status = -1; int DeviceNum = 0; void* Device = NULL; BootProfile_TypeDef BootProfile; BootProfile.DevicePowerSupply = USBPortAndPowerPort; BootProfile.PhysicalInterface = USB; BootInfo_TypeDef BootInfo; Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);	

uint64_t UID_L64; uint32_t UID_H32;

Status = Device_GetFullUID(&Device, & UID_L64, & UID_H32);

Status = Device_Close(&Device);

8.10 Device_GetHardwareState

```
int Device_GetHardwareState(
    void** Device,
    HardWareState_TypeDef* HardWareState
)
```

功能描述

获取设备硬件状态信息，包括 GNSS 外设类型、接收机类型、OCXO 类型，以及内部时钟、信号源、ADC 可变采样率和中频滤波器等支持情况。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] **Device**	设备句柄。
[out] **HardWareState**	输出设备硬件状态信息，包括 GNSS 外设类型、信号源支持、ADC 可变采样率及其他功能状态。 详见 HardWareState_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

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

BootProfile_TypeDef BootProfile;

BootProfile.DevicePowerSupply = USBPortAndPowerPort;

BootProfile.PhysicalInterface = USB;

BootInfo_TypeDef BootInfo;

Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);

HardWareState_TypeDef HardWareState;

Status = Device_GetHardwareState (&Device, & HardWareState);

Status = Device_Close(&Device);

8.11 Device_QueryDeviceInfoWithBus

```
int Device_QueryDeviceInfoWithBus(
    int DeviceNum,
    const BootProfile_TypeDef* BootProfile,
    BootInfo_TypeDef* BootInfo
)
```

功能描述

通过设备号查询设备信息，并返回设备基本信息、总线速度与版本、API 版本，以及启动过程中的错误和警告。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] DeviceNum	指定要打开的设备编号。
[in] BootProfile	设备启动配置，包含接口类型、供电方式及网络参数，用于初始化设备。 详见 BootProfile_TypeDef 结构体的定义。
[out] BootInfo	返回设备启动信息，包括设备信息、总线速度与版本、API 版本，以及启动过程中的错误和警告。 详见 BootInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	调用此函数前，设备必须处于未打开状态，即需要在 Device_Open 之前调用。

示例

```
int Status = -1; int DeviceNum = 0;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_QueryDeviceInfoWithBus(DeviceNum, &BootProfile, &BootInfo);
```

8.12 Device_SetFreqScan

```
int Device_SetFreqScan(
    void** Device,
    double StartFreq_Hz,
    double StopFreq_Hz,
    uint16_t SweepPts
)
```

功能描述	
配置设备的频率扫描参数，包括起始频率、终止频率及扫描点数，用于频谱扫描设置。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] StartFreq_Hz	扫描起始频率，单位 Hz。
[in] StopFreq_Hz	扫描终止频率，单位 Hz。
[in] SweepPts	扫描点数，即需要扫描的频点数量。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
double StartFreq_Hz = 1e9;
double StopFreq_Hz = 2e9;
uint16_t SweepPts = 5;
Status = Device_SetFreqScan(&Device, StartFreq_Hz, StopFreq_Hz, SweepPts);
Status = Device_Close(&Device);
```

8.13 Device_GetFreqPlanning

```
void Device_GetFreqPlanning(
    void** Device,
    double rf,
    uint8_t* rfb,
    uint8_t* convert,
    double* if1,
    double* im,
    double* rflo,
    double* iflo,
    double* if2
)
```

功能描述

根据输入的射频目标频率（RF），计算并返回系统内部的射频前端频率规划方案，包括射频频段、变频策略、第一/第二中频以及相应的本振频率。

兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] rf	输入目标射频频率，单位 Hz。
[out] rfb	目标射频频率所在的射频频段。
[out] convert	变频策略：上变频或下变频。
[out] if1	返回第一中频，单位 Hz。
[out] im	返回镜像频率，单位 Hz。
[out] rflo	返回射频本振频率，单位 Hz。
[out] iflo	返回中频本振频率，单位 Hz。
[out] if2	返回第二中频，单位 Hz。
返回值	无。
调用约束	需要在 Device_Open 后调用此函数。

示例

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

```
double rf = 1e9;
uint8_t rfb = 0, convert = 0; double if1 = 0, im = 0, rflo = 0, iflo = 0, if2 = 0;
Device_GetFreqPlanning(&Device, rf, &rfb, &convert, &if1, &im, &rflo, &iflo, &if2);
Status = Device_Close(&Device);
```

8.14 Device_SetIFOutput

```
int Device_SetIFOutput(
    void** Device,
    uint8_t* state,
)
```

功能描述	
中频输出控制。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] **Device**	设备句柄。
[in] **state**	控制中频输出。 0：关闭中频输出；1：开启中频输出。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
uint8_t state = 1;
Status = Device_SetIFOutput(&Device, state);
Status = Device_Close(&Device);
```

8.15 Device_QueryVersion_PMU

int Device_QueryVersion_PMU(void** Device, uint32_t* Version)	
功能描述	
查询设备电源管理单元（PMU）的版本号。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] Version	当前 PMU 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]表示修订版本。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
uint32_t PMUVersion = 0;
Status = Device_QueryVersion_PMU(&Device, &PMUVersion);
Status = Device_Close(&Device);	

8.16 Device_QueryVersion_AGU

int Device_QueryVersion_AGU(void** Device, uint32_t* Version)	
功能描述	
查询 AGU 的版本号。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。

[out] Version	当前 AGU 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]表示修订版本。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
uint32_t AGUVersion = 0;
Status = Device_QueryVersion_AGU(&Device, &AGUVersion);
Status = Device_Close(&Device);	

8.17 Device_InitIFAGC

int Device_InitIFAGC(void** Device)	
功能描述	
初始化 IFAGC，将 AGC 初始化为默认值。	
兼容性	0.55.79 及之后版本支持
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	请参考 Device_SetIFAGCTarget()函数的相关示例。

8.18 Device_SetIFAGCTarget

int Device_SetIFAGCTarget(void** Device, double* Target)	
功能描述	
设置 IFAGC 的目标功率。	
兼容性	0.55.79 及之后版本支持
参数说明	

[in] Device	设备句柄。
[in/out] Target	设置目标功率值，为距 ADC 饱和的 dBFs 值。 范围：0~-30 dBFs。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
Status = Device_InitIFAGC(&Device);
double Target = 10;
Status = Device_SetIFAGCTarget(&Device, &Target);
Status = Device_Close(&Device);
```

8.19 Device_ConvertEpochToReadable

```
void Device_ConvertEpochToReadable(
    uint64_t nsSinceEpoch,
    int16_t* year,
    int16_t* mon,
    int16_t* day,
    int16_t* hour,
    int16_t* min,
    int16_t* sec,
    int16_t* ms,
    int16_t* us,
    int16_t* ns
)
```

功能描述	
将 ns 级纪元 nsSinceEpoch 转为物理时间（UTC 时间）。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] nsSinceEpoch	输入当前数据包中所对应的 ns 级时间戳。
[out] year	返回年份。
[out] mon	返回月份。

[out] day	返回日期。
[out] hour	返回小时。
[out] min	返回分钟。
[out] sec	返回秒数。
[out] ms	返回毫秒数。
[out] us	返回微秒数。
[out] ns	返回纳秒数。
返回值	无。
调用约束	需要在 GNSS 锁定并调用 Get 函数获取到数据后调用此函数。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
GNSSAntennaState_TypeDef GNSSAntennaState = GNSS_AntennaExternal;
Status = Device_SetGNSSAntennaState(&Device, GNSSAntennaState);
GNSSInfo_TypeDef GNSSInfo = { 0 };
while (!GNSSInfo.GNSS_LockState) {
    Status = Device_GetGNSSInfo_Realtime(&Device, &GNSSInfo);
    this_thread::sleep_for(chrono::milliseconds(1000));
}
SWP_Profile_TypeDef SWP_ProfileIn, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.PartialsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.PartialsweepTracePoints);
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency.data(), PowerSpec_dBm.data(),
&HopIndex, &FrameIndex, &MeasAuxInfo);
```

```
int16_t year = 0, mon = 0, day = 0, hour = 0, min = 0, sec = 0, ms = 0, us = 0, ns = 0;
Device_ConvertEpochToReadable(MeasAuxInfo.nsSinceEpoch, &year, &mon, &day, &hour,
&min, &sec, &ms, &us, &ns);
Status = Device_Close(&Device);
```

8.20 Device_GetAmpAttenState

```
void Device_GetAmpAttenState(
    void** Device,
    PreamplifierState_TypeDef* amp,
    int8_t* att,
    uint8_t* sp
)
```

功能描述	
查询增益映射参数。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] amp	返回预选放大器状态。 详见 PreamplifierState_TypeDef 结构体的定义。
[out] att	返回频谱仪通道衰减量，单位 dB。
[out] sp	返回增益空间值，内部参数，用户无需关注或使用。
返回值	无。
调用约束	需要在 xxx_Configuration 后调用此函数。

示例

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn;
SWP_Profile_TypeDef SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.Atten = 3;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
```

```
PreamplifierState_TypeDef ampState; int8_t att; uint8_t sp;
Device_GetAmpAttenState(&Device, &ampState, &att, &sp);
Status = Device_Close(&Device);
```

8.21 Device_QueryEIO_Version_UID

```
int Device_QueryEIO_Version_UID(
    void** Device,
    uint16_t* EIOVersion,
    uint64_t* EIOUID
)
```

功能描述

获取与指定设备相连的 GNSS 模块的版本与 UID 号。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] EIOVersion	返回 GNSS 模块的版本号。采用主版本，修订版本表示。 bit[15..8]表示主版本，bit[7..0]表示修订版本。
[out] EIOUID	返回 GNSS 模块的 UID 号。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在调用 Device_Open 后使用此函数。

示例

```
int Status = 0; int DevNum = 0; void* Device = NULL;
uint16_t EIOVersion = 0; uint64_t EIOUID = 0;
int major = 0, minor = 0, rev = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
Status = Device_QueryEIO_Version_UID(&Device, &EIOVersion, &EIOUID);
major = (EIOVersion >> 16) & 0xffff;
minor = (EIOVersion >> 8) & 0xff;
rev = EIOVersion & 0xff;
Status = Device_Close(&Device);
```

8.22 Device_GPIOSetBits

int Device_GPIOSetBits(void** Device, uint16_t Bits)	
功能描述	
EIO 选件 GPIO 引脚拉高。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] Bits	GPIO 控制位。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	请参考 Device_GPIOBandSwitch()函数的相关示例。

8.23 Device_GPIOResetBits

int Device_GPIOResetBits(void** Device, uint16_t Bits)	
功能描述	
EIO 选件 GPIO 引脚拉低。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] Bits	GPIO 控制位。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	请参考 Device_GPIOBandSwitch()函数的相关示例。

8.24 Device_GPIOBandSwitch

```
int Device_GPIOBandSwitch(
    void** Device,
    uint16_t Bands,
    double StartFreq[],
    double StopFreq[],
    uint16_t SetBits[],
    uint16_t ResetBits[],
    uint32_t delay_us[]
)
```

功能描述	
EIO 选件 GPIO 引脚跟随频率分段变化。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] Bands	频段数量。
[in] StartFreq[]	起始频率，单位为 Hz。
[in] StopFreq[]	终止频率，单位为 Hz。
[in] SetBits[]	拉高 GPIO 引脚。
[in] ResetBits[]	拉低 GPIO 引脚。
[in] delay_us[]	频段切换延迟。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例

```
int Status = 0; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
std::cout << "Device_Open: " << Status << std::endl;
if (Status) {
    return 0;
}
// set the number of frequency bands
```

```
uint16_t Bands = 3;
// band 0: 1e9 to 1.5e9
// band 1: 3e9 to 5e9
// band 2: 7e9 to 8e9
double StartFreq[] = { 1e9,3e9,7e9 };
double StopFreq[] = { 1.5e9,5e9,8e9 };
// set GPIO
// pull up GPIO 0, 1, 2, all
// Device_GPIOSetBits(&Device, 0x01)
// Device_GPIOSetBits(&Device, 0x02)
// Device_GPIOSetBits(&Device, 0x04)
// Device_GPIOSetBits(&Device, 0xFF)
uint16_t setBits[] = { 0x01,0x02,0x04 };
// pull down GPIO 1&2
// Device_GPIOResetBits(&Device, 0x06)
uint16_t resetBits[] = { 0x06,0x05,0x03 };
uint32_t delays[] = { 100,100,100 };
Status = Device_GPIOBandSwitch(&Device, Bands, StartFreq, StopFreq, setBits, resetBits,
delays);
Status = Device_Close(&Device);
```

9. 系统 Device 与 GNSS 相关函数

9.1 Device_SetGNSSAntennaState

```
int Device_SetGNSSAntennaState(
    void** Device,
    const GNSSAntennaState_TypeDef GNSSAntennaState
)
```

功能描述

设置 GNSS 天线状态，用于控制天线开关或工作模式（需选件支持）。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] GNSSAntennaState	设置 GNSS 天线状态。 详见 GNSSAntennaState_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	请参考 Device_GetGNSSAntennaState()函数相关示例。

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

9.2 Device_SetGNSSxpps

```
int Device_SetGNSSxpps(
    void** device,
    uint8_t enableout,
    double xpps,
    double delay
)
```

功能描述

设定 GNSS 的秒脉冲。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] enableout	启用/禁用 GNSS 的秒脉冲。1：开启；0：关闭。
[in] xpps	设置秒脉冲的周期。范围：0.25~10^7。
[in] delay	设置 GNSS 秒脉冲的延迟，GNSS 锁定后准确度更高。范围：0~1/xpps。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	

int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
uint8_t Enableout = 1;double xpps= 1; double delay =0;
Status = Device_SetGNSSxpps (&Device, Enableout, xpps, delay);
Status = Device_Close(&Device);	

9.3 Device_SetDOCXOWorkMode

int Device_SetDOCXOWorkMode(void** Device, const DOCXOWorkMode_TypeDef DOCXOWorkMode)	
功能描述	
在使用 GNSS 功能时，设置 DOCXO 的工作状态。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] DOCXOWorkMode	设置 DOCXO 工作模式。 详见 DOCXOWorkMode_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。

示例	
int Status = -1; int DeviceNum = 0; void* Device = NULL; BootProfile_TypeDef BootProfile; BootProfile.DevicePowerSupply = USBPortAndPowerPort; BootProfile.PhysicalInterface = USB; BootInfo_TypeDef BootInfo; Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo); DOCXOWorkMode_TypeDef DOCXOWorkMode = DOCXO_LockMode; Status = Device_SetDOCXOWorkMode(&Device, DOCXOWorkMode); Status = Device_Close(&Device);	

9.4 Device_GetGNSSInfo

int Device_GetGNSSInfo(void** Device, GNSSInfo_TypeDef* GNSSInfo)	
功能描述	
在使用 GNSS 功能时，获取 GNSS 设备状态信息（需选件支持）。 Device_GetGNSSInfo：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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] GNSSInfo	返回 GNSS 设备状态信息，包括定位坐标、海拔、卫星数量、GNSS 与 DOCXO 状态、天线状态等。 详见 GNSSInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	
int Status = -1; int DeviceNum = 0; void* Device = NULL; BootProfile_TypeDef BootProfile; BootProfile.DevicePowerSupply = USBPortAndPowerPort; BootProfile.PhysicalInterface = USB; BootInfo_TypeDef BootInfo; Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);	

```
Status = Device_SetGNSSAntennaState(&Device, GNSS_AntennaExternal);
GNSSInfo_TypeDef GNSSInfo;
Status = Device_GetGNSSInfo(&Device, & GNSSInfo);
Status = Device_Close(&Device);
```

9.5 Device_GetGNSS_SatDate/Device_GetGNSS_SatDate_Realtime

```
int Device_GetGNSS_SatDate(
    void** Device,
    GNSS_SatDate _TypeDef* GNSS_SatDate
)
```

```
int Device_GetGNSS_SatDate_Realtime(
    void** Device,
    GNSS_SatDate_TypeDef* GNSS_SatDate
)
```

功能描述

获取设备当前的 GNSS 卫星信噪比信息（需要选件支持），包括可视卫星数量和用于定位的卫星信号强度。

Device_GetGNSS_SatDate：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。

Device_GetGNSS_SatDate_Realtime：实时获取，短时间内会占用数据通道。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] GNSS_SatDate	返回当前设备的 GNSS 卫星信息，包括可视卫星数量和用于定位的卫星信息。 详见 GNSS_SatDate_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。 注意：仅在 GNSS 锁定后，获取的信噪比和卫星数量才是有效值，默认是 0。

示例

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

```
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
Status = Device_SetGNSSAntennaState(&Device, GNSS_AntennaExternal);
GNSS_SatDate_TypeDef GNSS_SatDate;
Status = Device_GetGNSS_SatDate(&Device, &GNSS_SatDate);
Status = Device_GetGNSS_SatDate_Realtime(&Device, &GNSS_SatDate);
Status = Device_Close(&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	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UserProfile_O	输出默认的配置参数集合。 详见 SWP_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 Device_Open 后调用此函数。
示例	请参考 SWP_GetPartialSweep()函数相关示例。

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	
将频谱仪设备配置成标准频谱分析（SWP）模式，并配置该模式下相关参数。 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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] ProfileIn	配置集输入。

	详见 SWP_Profile_TypeDef 结构体的定义。
[out] ProfileOut	配置集输出，并非所有 ProfileIn 中数据都能被严格采纳，设备会根据硬件能力或内部约束自动调整部分参数，实际下发以配置集输出为准。 详见 SWP_Profile_TypeDef 结构体的定义。
[out] TraceInfo	返回频谱迹线的相关信息。 详见 SWP_TraceInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_ProfileDeInit 之后进行调用。
示例	请参考 SWP_GetPartialSweep()函数相关示例。

10.3 SWP_AutoSet

```
int SWP_AutoSet(
    void** Device,
    SWPApplication_TypeDef Application,
    const SWP_Profile_TypeDef* ProfileIn,
    SWP_Profile_TypeDef* ProfileOut,
    SWP_TraceInfo_TypeDef* TraceInfo,
    uint8_t ifDoConfig
)
```

功能描述	
标准频谱分析模式下，根据应用目标给出推荐设备配置。 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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] Application	设定测量类型。 详见 SWPApplication_TypeDef 枚举的定义。
[in] ProfileIn	配置集输入。 详见 SWP_Profile_TypeDef 结构体的定义。
[out] ProfileOut	配置集输出。 详见 SWP_Profile_TypeDef 结构体的定义。
[out] TraceInfo	返回频谱迹线的相关信息。 详见 SWP_TraceInfo_TypeDef 结构体的定义。
[in] ifDoConfig	设置是否需要单独调用SWP_Configuration()函数。

	0：需要；1：不需要。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	当 ifDoConfig 值为 0 时，需要在 SWP_Configuration 之前调用； 当 ifDoConfig 值为 1 时，函数内部会自己调用 SWP_Configuration 函数，因此不需要额外调用 SWP_Configuration。
示例：（ifDoConfig 值为 1）	

int Status = -1;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
uint8_t ifDoConfig=1;
SWPApplication_TypeDef Application;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_AutoSet (&Device, Application ,&ProfileIn, &ProfileOut, &TraceInfo, ifDoConfig);
 	
Status = Device_Close(&Device);	

10.4 SWP_GetPartialSweep

int SWP_GetPartialSweep(void** Device, double Freq_Hz[], float PowerSpec_dBm[], int* HopIndex, int* FrameIndex, MeasAuxInfo_TypeDef* MeasAuxInfo)
功能描述
获取 SWP 模式下在每个跳频点上获取的频谱结果，同时返回此时跳频频点序号、帧序号和测量数据的辅助信息，以便用户自行将多次扫描的结果拼接成整个频谱曲线，并返回函数状态。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。

[out] Freq_Hz[]	返回当前跳频点对应的频率数组，单位为 Hz。 数组大小等于 TraceInfo.PartialSweepTracePoints。
[out] PowerSpec_dBm[]	返回当前跳频点对应的幅度数组，单位为 dBm。 数组大小等于 TraceInfo.PartialSweepTracePoints。
[out] HopIndex	返回数据的跳频频点序号。
[out] FrameIndex	返回数据的帧序号，仅扫描时间模式非 SweepTimeMode=SWTMode_minSWT，且检波器 Detector=MaxPower 生效。
[out] MeasAuxInfo	返回数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn,SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.StartFreq_Hz = 9e3;
SWP_ProfileIn.StopFreq_Hz = 6.35e9;
SWP_ProfileIn.RBWMode = RBW_Manual;
SWP_ProfileIn.RBW_Hz = 200e3;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
int HopIndex = 0, FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
for (int i = 0; i < TraceInfo.TotalHops; i++) {
Status = SWP_GetPartialSweep(&Device, Frequency.data() + i * TraceInfo.PartialsweepTracePoints, PowerSpec_dBm.data() + i * TraceInfo.PartialsweepTracePoints, &HopIndex, &FrameIndex, &MeasAuxInfo);
}

Device_Close(&Device);
```

10.5 SWP_GetFullSweep

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

功能描述

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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] Freq_Hz[]	返回的频率数组，单位为 Hz。 数组大小等于 TraceInfo.PartialSweepTracePoints。
[out] PowerSpec_dBm[]	返回的幅度数组，单位为 dBm。 数组大小等于 TraceInfo.PartialSweepTracePoints。
[out] MeasAuxInfo	返回数据的辅助信息。详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn, ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &Mea
sAuxInfo);

Status = Device_Close(&Device);
```

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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] PartialTrace	返回包含配置、返回信息和暂存数据的顶层结构体。 详见 SWPTrace_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &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;
vector<double> Frequency(TraceInfo.PartialsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.PartialsweepTracePoints);
PartialTrace.Freq_Hz = Frequency.data();
PartialTrace.PowerSpec_dBm = PowerSpec_dBm.data();
Status = SWP_GetPartialSweep_PM1(&Device, &PartialTrace);
Status = Device_Close(&Device);
```

11.2 SWP_ResetTraceHold

void SWP_ResetTraceHold(void** Device)	
功能描述	
迹线更新方式 SWP_TraceType_TypeDef 设置为 MaxHold 或 MinHold 时，重置保持的迹线数据。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	无。
调用约束	仅在 SWP_TraceType_TypeDef 为 MaxHold 或 MinHold 时生效。 需要在 SWP_Get 函数之后进行调用。
示例	

```
int Status = -1;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &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);
vector<double> Frequency(TraceInfo.PartialsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.PartialsweepTracePoints);
int HopIndex = 0;int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetPartialSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &H
opIndex,&FrameIndex, &MeasAuxInfo);
SWP_ResetTraceHold(&Device);
Status = Device_Close(&Device);
```

12. 相位噪声测量模式

12.1 PNM_ProfileDeInit

int PNM_ProfileDeInit(void** Device, PNM_Profile_TypeDef* PNM_Profile)	
功能描述	
提供相位噪声测量的默认配置参数，初始化设备的相位噪声测量设置。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] PNM_Profile	相位噪声测量的配置结构体。 详见 PNM_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常。
调用约束	在 Device_Open 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.2 PNM_Configuration

int PNM_Configuration (void** Device, const PNM_Profile_TypeDef* PNM_Profile_in, PNM_Profile_TypeDef* PNM_Profile_out, PNM_MeasInfo_TypeDef* PNM_MeasInfo)	
功能描述	
配置相位噪声测量相关参数。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] PNM_Profile_in	配置输入结构体。 详见 PNM_Profile_TypeDef 结构体的定义。

[out] PNM_Profile_out	配置输出结构体，返回设备实际生效的测量参数。 详见 PNM_Profile_TypeDef 结构体的定义。
[out] PNM_MeasInfo	相位噪声测量信息结构体。 详见 PNM_MeasInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_ProfileDeInit 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.3 PNM_StartMeasure

int PNM_StartMeasure(void** Device)	
功能描述	
开始一次相位噪声测量，测量开始前，调用此函数启动测量。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_Configuration 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.4 PNM_StopMeasure

int PNM_StopMeasure(void** Device)	
功能描述	
强制停止本次相位噪声测量。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_StartMeasure 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.5 PNM_GetPartialUpdatedFullTrace

```
int PNM_GetPartialUpdatedFullTrace(
        void** Device,
        double* CarrierFreq,
        float* CarrierPower,
        double Freq[],
        float PhaseNoise[],
        uint32_t FrameUpdateCounts[],
        MeasAuxInfo_TypeDef* MeasAuxInfo,
        float* RefLevel
)
```

功能描述	
获取相位噪声测量结果迹线。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] CarrierFreq	载波频率。
[out] CarrierPower	载波功率。
[out] Freq[]	迹线频率轴，单位 Hz。
[out] PhaseNoise[]	迹线功率轴，单位 dBc/Hz。
[out] FrameUpdateCounts[]	刷新计数器（索引 0 为最远端的分段，N 为最近端的分段；元素为段的已刷新帧数）。
[out] MeasAuxInfo	辅助测量信息结构体。 详见MeasAuxInfo_TypeDef结构体的定义。
[out] RefLevel	返回当前测量对应的参考电平。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_StartMeasure 之后进行调用，调用 PartialUpdateCounts 次 Get 接口可获得一次完整测量结果。之后若继续测量，需要重新调用 StartMeasure 接口。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.6 PNM_AutoSearch

int PNM_AutoSearch (void** Device, double* CarrierFreq, uint8_t Found)	
功能描述	
通过全景扫描寻找功率超过载波门限的信号。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] CarrierFreq	载波频率。
[out] Found	载波存在标志，0：无载波；1：有载波。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_Configuration 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.7 PNM_Preset_FrameDetRatio

int PNM_Preset_FrameDetRatio (void** Device, PNM_MeasInfo_TypeDef* MeasInfo)	
功能描述	
复位各频率分段的帧检波比。	
兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] MeasInfo	复位帧检波比后，更新相位噪声测量的测量信息结构体 详见 PNM_MeasInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_Configuration 之后进行调用。
示例	请参考 PNM_Set_FrameDetRatio()函数相关示例。

12.8 PNM_Set_FrameDetRatio

```
int PNM_Set_FrameDetRatio (
        void** Device,
        uint32_t FrameDetRatioOfSegment[],
        PNM_MeasInfo_TypeDef* MeasInfo
)
```

功能描述

手动配置各频率分段的帧检波比。

兼容性	0.55.58 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] FrameDetRatioOfSegment[]	帧检波比的设定数组（数组长度为 Segments；索引 0 为最远端的分段，N 为最近端的分段）
[out] MeasInfo	调整帧检波比后，更新相位噪声测量的测量信息结构体 详见 PNM_MeasInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 PNM_Configuration 之后进行调用。

示例

```
int Status = 0;void* Device = NULL;int DeviceNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
PNM_Profile_TypeDef PNM_ProfileIn, PNM_ProfileOut;
PNM_MeasInfo_TypeDef PNM_MeasInfo;
PNM_ProfileDeInit(&Device, &PNM_ProfileIn);
PNM_ProfileIn.CenterFreq = 1e9;
PNM_ProfileIn.Threshold = -50.0;
PNM_ProfileIn.RBWRatio = 0.1;
PNM_ProfileIn.StartOffsetFreq = 100;
PNM_ProfileIn.StopOffsetFreq = 1e6;
PNM_ProfileIn.TraceAverage = 1;
```

```
// 配置设备的相位噪声测量状态
Status = PNM_Configuration(&Device, &PNM_ProfileIn, &PNM_ProfileOut, &PNM_MeasInfo);
// 全频段寻载波（高级功能非必需）：可使用 PNM_AutoSearch 接口进行全景扫描，寻找超过载波判决门限的信号
//double PeakFreq = 1.0;
//uint8_t Found = 0;
//Status = PNM_AutoSearch(&Device, &PeakFreq, &Found);
// 若寻到理想载波，可将该频率通过 PNM_Configuration 接口配置给设备进行分析。
// 手动设定检波比（高级功能非必需）：可使用 PNM_Set_FrameDetRatio 接口手动调整各频率分段的帧检波比
// PNM_MeasInfo.FrameDetRatioOfSegment[PNM_MeasInfo.Segments - 1] = 10;
// Status = PNM_Set_FrameDetRatio(&Device, PNM_MeasInfo.FrameDetRatioOfSegment, &PNM_MeasInfo);
// 复位帧检波比（高级功能非必需）：可使用 PNM_Preset_FrameDetRatio 接口将各段的帧检波比复位为默认值
// PNM_Preset_FrameDetRatio(&Device, &PNM_MeasInfo);
vector<double> Freq(PNM_MeasInfo.TracePoints);
vector<float> PhaseNoise(PNM_MeasInfo.TracePoints);
double CarrierFreq;float CarrierPower;float RefLevel;
vector<uint32_t> FrameUpdateCounts(PNM_MeasInfo.Segments);
MeasAuxInfo_TypeDef MeasAuxInfo;
while(1)
{
PNM_StartMeasure(&Device); // 开始一次相位噪声测量
for (int i = 0; i < PNM_MeasInfo.PartialUpdateCounts; i++) // 获取一次相位噪声测量结果的迹线
{
Status = PNM_GetPartialUpdatedFullTrace(&Device, &CarrierFreq, &CarrierPower, Freq.data(), PhaseNoise.data(), FrameUpdateCounts.data(), &MeasAuxInfo, &RefLevel);
if (Status == APIRETVAL_NoError)
{
}
else
{
switch (Status)
{
```

```
case APIRETVAL_WARNING_CarrierLoss :
{
cout << "未发现载波" << endl; //未在指定频点附近未找到功率高于载波判决门限的信号
break;
}
case APIRETVAL_WARNING_MeasUpdate :
{
i = 0;
cout << "测量状态更新" << endl; // PNM 特殊返回值：相位噪声测量过程中，DUT 状态发生改变，导致测量状态自动更新，需重新获取 PartialUpdateCounts 次数据
break;
}
default: cout << "请对应编程指南检查通用错误码" << endl;
}
}
}
cout << "wait";
}
PNM_StopMeasure(&Device); //  停止本次相位噪声测量
Device_Close(&Device);
```

13. 接收机/IQ 流 IQS 主要函数

13.1 IQS_ProfileDeInit

int IQS_ProfileDeInit(void** Device, IQS_Profile_TypeDef* UserProfile_O)	
功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UserProfile_O	IQS 配置结构体指针。 函数执行后，该结构体将被填充为系统定义的默认配置值。 详见 IQS_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 后调用。
示例	请参考 IQS_GetIQStream()函数相关示例。

13.2 IQS_Configuration

int IQS_Configuration(void** Device, const IQS_Profile_TypeDef* ProfileIn, IQS_Profile_TypeDef* ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)	
功能描述	
将频谱仪配置成接收机/IQ 流(IQS)模式。 IQS 模式下，本振信号固定，并接收以本振频率为中心，具有一定带宽的射频信号。 当抽取倍数≥2 时，使用 USB3.0 和高速硬盘进行数据传输可实现连续时域记录功能。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。

[in] IQS_ProfileIn	IQS 配置结构体指针。 详见 IQS_Profile_TypeDef 结构体的定义。
[out] IQS_ProfileOut	IQS 配置结构体指针。并非所有 ProfileIn 中数据都能被严格采纳，设备会根据硬件能力或内部约束自动调整部分参数，实际下发以配置集输出为准。 详见 IQS_Profile_TypeDef 结构体的定义。
[out] StreamInfo	IQ 流模式下，IQ 时域数据流的相关信息。 详见 IQS_StreamInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 IQS_ProfileDeInit 之后进行调用。
示例	请参考 IQS_GetIQStream()函数相关示例。

13.3 IQS_BusTriggerStart

int IQS_BusTriggerStart(void** Device)	
功能描述	
将发起总线触发。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 IQS_GetIQStream 之前进行调用。
示例	请参考 IQS_GetIQStream()函数相关示例。

13.4 IQS_BusTriggerStop

int IQS_BusTriggerStop(void** Device)	
功能描述	
终止当前的总线触发。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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 IQS_GetIQStream 之前进行调用。
示例	请参考 IQS_GetIQStream()函数相关示例。

13.5 IQS_GetIQStream

```
int 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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] 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。 IQ 数据类型设置为 int8_t 时，I、Q 两路分别为 32484 个点，每个点占 1 个字节； IQ 数据类型设置为 int16_t 时，I、Q 两路分别为 16242 个点，每个点占 2 个字节； IQ 数据类型设置为 int32_t 时，I、Q 两路分别为 8121 个点，每个点占 4 个字节。 IQ 数据按照 IQIQ...的排列方式存储。
[out] ScaleToV	将 ADC 采集的原始时域数据转换为电压绝对值（V）的系数。
[out] TriggerInfo	IQ 数据流的触发相关信息。 详见 IQS_TriggerInfo_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
```

```
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_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;
Status = IQS_GetIQStream(&Device, &AlternIQStream, &ScaleToV, &TriggerInfo,
&MeasAuxInfo);
Status = IQS_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

14. 接收机/IQ 流 IQS 其他函数

14.1 IQS_MultiDevice_WaitExternalSync

int IQS_MultiDevice_WaitExternalSync(void** Device, const IQS_Profile_TypeDef* ProfileIn)	
功能描述	
调用该函数等待多机同步触发信号，以便多个设备能够在同步时刻启动或进行数据采集。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] ProfileIn	IQS 配置结构体指针。 详见 IQS_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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	请参考 IQS_MultiDevice_Run()函数相关示例。

14.2 IQS_MultiDevice_Run

int IQS_MultiDevice_Run(void** Device)	
功能描述	
调用该函数使能多设备同步运行，启动多个设备以进行同步的数据采集或操作。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 IQS_MultiDevice_WaitExternalSync 之后进行调用。
示例	

int Status = -1, Status0 = -1; int DeviceNum0 = 0; int DeviceNum1 = 0;
void* Device0 = NULL; void* Device1 = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;	

```
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device0, DeviceNum0, &BootProfile, &BootInfo);
Status0 = Device_Open(&Device1, DeviceNum1, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn0, ProfileIn1;
IQS_Profile_TypeDef ProfileOut0, ProfileOut1;
IQS_StreamInfo_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);
Status0 = IQS_Configuration(&Device1, &ProfileIn1, &ProfileOut1, &StreamInfo1);
Status0 = IQS_MultiDevice_WaitExternalSync(&Device0, &ProfileOut0);
Status0 = IQS_MultiDevice_Run(&Device0);
Status = IQS_MultiDevice_Run(&Device1);
Status = Device_Close(&Device0);
Status0 = Device_Close(&Device1);
```

14.3 IQS_SyncTimer

int IQS_SyncTimer(void** Device)	
功能描述	
调用此函数发起定时器-外触发同步。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
```

```
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
ProfileIn. TriggerTimerSync = SyncToExt_RisingEdge;
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_SyncTimer (&Device);
Status = Device_Close(&Device);
```

14.4 IQS_GetIQStream_PM1

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

功能描述

获取 IQS 模式下的时域数据，时域数据格式为 int8_t、int16_t 或 int32_t，用户可根据实际需求进行选择。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] IQStream	IQ 数据流，包括 IQ 数据及相关配置信息等。 详见 IQStream_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
```

```
IQS_StreamInfo_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);
Status = IQS_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

14.5 IQS_GetIQStream_PM2

```
int IQS_GetIQStream_PM2(
    void** Device,
    IQStream_TypeDef* IQStream,
    MeasAuxInfo_TypeDef* MeasAuxInfo
)
```

功能描述

获取 IQS 模式下的时域数据和测量辅助信息，时域数据格式为 int8_t、int16_t 和 int32_t，用户可根据需要选择数据格式。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] IQStream	IQ 数据流，包括 IQ 数据及相关配置信息等。 详见 IQStream_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
```

```
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM2(&Device, &IQStream, &MeasAuxInfo);
Status = IQS_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

14.6 IQS_GetIQStream_Data

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

功能描述

调用此函数获取 int16_t 类型的 IQ 时域数据。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
```

```
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
vector<int16_t> IQ_Data(StreamInfo.StreamSamples);
Status = IQS_GetIQStream_Data(&Device, IQ_Data.data());
Status = Device_Close(&Device);
```

15. 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。

15.1 DET_ProfileDeInit

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

功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UserProfile_O	DET 配置结构体指针。 详见 DET_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 后调用。
示例	请参考 DET_GetPowerStream()函数相关示例。

15.2 DET_Configuration

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

功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。

[in] DET_ProfileIn	DET 配置结构体指针。 详见 DET_Profile_TypeDef 结构体的定义。
[out] DET_ProfileOut	DET 配置结构体指针。 详见 DET_Profile_TypeDef 结构体的定义。
[out] StreamInfo	DET 模式下，DET 数据的相关信息。 详见 DET_StreamInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DET_ProfileDeInit 之后进行调用。
示例	请参考 DET_GetPowerStream()函数相关示例。

15.3 DET_BusTriggerStart

int DET_BusTriggerStart(void** Device)	
功能描述	
将发起总线触发。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DET_GetPowerStream 之前进行调用。
示例	请参考 DET_GetPowerStream()函数相关示例。

15.4 DET_BusTriggerStop

int DET_BusTriggerStop(void** Device)	
功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DET_GetPowerStream 之前进行调用。
示例	请参考 DET_GetPowerStream()函数相关示例。

15.5 DET_GetPowerStream

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

功能描述

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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] NormalizedPowerStream[]	返回每个数据点的瞬时幅值，计算方式为$\sqrt{I^2+Q^2}$。
[out] ScaleToV	将无单位的瞬时幅值（NormalizedPowerStream）转换为物理电压绝对值（V）的系数。
[out] TriggerInfo	DET 数据的触发相关信息。 详见 DET_TriggerInfo_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需在 DET_Configuration 之后进行调用。

示例

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

```
vector<float> NormalizedPowerStream(StreamInfo.PacketSamples);
float ScaleToV = 0;
DET_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = DET_BusTriggerStart(&Device);
Status = DET_GetPowerStream(&Device, NormalizedPowerStream.data(), &ScaleToV,
&TriggerInfo, &MeasAuxInfo);
Status = DET_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

15.6 DET_SyncTimer

int DET_SyncTimer(void** Device)	
功能描述	
调用此函数发起定时器-外触发同步。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
DET_Profile_TypeDef ProfileIn, ProfileOut;
DET_StreamInfo_TypeDef StreamInfo;
Status = DET_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
ProfileIn. TriggerTimerSync = SyncToExt_RisingEdge;
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_SyncTimer (&Device);
Status = Device_Close(&Device);
```

16. 零扫宽 ZeroSpan 模式

ZeroSpan模式用于对信号进行零扫宽分析，提供信号在特定频率下的实时功率测量，帮助用户精确捕捉瞬时信号变化。

16.1 ZSP_ProfileDeInit

```
int ZSP_ProfileDeInit(
    void** Device,
    ZSP_Profile_TypeDef* UserProfile_O
)
```

功能描述

初始化配置 ZeroSpan 模式的相关参数。

ZeroSpan 模式下的中心频率、参考电平、抽取倍数等参数统一封装在 ZSP_Profile_TypeDef 结构体中。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UserProfile_O	ZeroSpan 配置结构体指针。 详见 ZSP_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 后调用。
示例	请参考 ZSP_Configuration()函数相关示例。

16.2 ZSP_Configuration

```
int ZSP_Configuration(
    void** Device,
    const ZSP_Profile_TypeDef* ProfileIn,
    ZSP_Profile_TypeDef* ProfileOut,
    DET_StreamInfo_TypeDef* StreamInfo
)
```

功能描述

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

兼容性	0.55.77 及之后版本支持
参数说明	
[in] **Device**	设备句柄。
[in] **ProfileIn**	ZSP 配置结构体指针。 详见 ZSP_Profile_TypeDef 结构体的定义。
[out] **ProfileOut**	ZSP 配置结构体指针。 详见 ZSP_Profile_TypeDef 结构体的定义。
[out] **StreamInfo**	ZSP 模式下，ZSP 数据的相关信息。 详见 DET_StreamInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 ZSP_ProfileDeInit 之后进行调用。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
ZSP_Profile_TypeDef ProfileIn, ProfileOut;
DET_StreamInfo_TypeDef StreamInfo;
Status = ZSP_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.CenterFreq_Hz = 1e9;
ProfileIn.DecimateFactor = 2;
Status = ZSP_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = Device_Close(&Device);
```

17. 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。

17.1 RTA_ProfileDeInit

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

功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] UserProfile_O	RTA 配置结构体指针。 详见 RTA_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 后调用。
示例	请参考 RTA_GetRealTimeSpectrum()函数相关示例。

17.2 RTA_Configuration

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

功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。

[in] RTA_ProfileIn	RTA 配置结构体指针。 详见 RTA_Profile_TypeDef 结构体的定义。
[out] RTA_ProfileOut	RTA 配置结构体指针。 详见 RTA_Profile_TypeDef 结构体的定义。
[out] FrameInfo	RTA 模式下，配置后返回的包信息结构体。 详见 RTA_FrameInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 RTA_ProfileDeInit 后调用。
示例	请参考 RTA_GetRealTimeSpectrum()函数相关示例。

17.3 RTA_BusTriggerStart

int RTA_BusTriggerStart(void** Device)	
功能描述	
将发起总线触发。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 RTA_GetRealTimeSpectrum 之前进行调用。
示例	请参考 RTA_GetRealTimeSpectrum()函数相关示例。

17.4 RTA_BusTriggerStop

int RTA_BusTriggerStop(void** Device)	
功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 RTA_GetRealTimeSpectrum 之后进行调用。
示例	请参考 RTA_GetRealTimeSpectrum()函数相关示例。

17.5 RTA_SetDataFormat

int RTA_SetDataFormat(void** Device, DataFormat_TypeDef* DataFormat)	
功能描述	
设置实时频谱的数据类型。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] DataFormat	设置实时频谱的数据类型。 详见 DataFormat_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 RTA_Configuration 之前调用。
示例	请参考 RTA_GetIQStream()函数相关示例。

17.6 RTA_SetLookBackCmd

int RTA_SetLookBackCmd(void** Device, LookBack_TypeDef* LookBackCmd)	
功能描述	
设置实时频谱 LookBack 状态，使能 LookBack 后，设备会缓存频谱对应的 IQ 数据，供用户在需要的情况下读取。	
兼容性	0.55. 79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] LookBackCmd	设置实时频谱 LookBack 状态。 详见 LookBack_TypeDef 枚举的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 RTA_Configuration 之前调用。
示例	请参考 RTA_GetIQStream()函数相关示例。

17.7 RTA_TriggerStart

int RTA_TriggerStart(void** Device)	
功能描述	
触发开始。当触发为总线触发时，调用该函数会立刻进行触发，当触发为非总线触发时，该函数会使能触发，并等待触发到来后触发设备采集。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 RTA_GetRealTimeSpectrum 之前进行调用。
示例	请参考 RTA_GetIQStream()函数相关示例。

17.8 RTA_GetIQStream

int RTA_GetIQStream(void** Device, void* AlternIQStream, float* ScaleToV, IQS_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)	
功能描述	
获取 RTA 模式下的实时 IQStream 及触发相关的信息。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] AlternIQStream	时域数据（IQ 数据以 IQIQ...的交织排列方式存储）。 一个数据包最多可以包含 8192 个 IQ 点，数据包的总字节数由 IQS_TriggerInfo_TypeDef 结构体中的 InPacketTriggeredDataSize 参数决定。 数据类型为 int16_t 时，总数据点数：InPacketTriggeredDataSize / 2; 数据类型为 int8_t 时，总数据点数：InPacketTriggeredDataSize; 数据类型为 int32_t 时，总数据点数 InPacketTriggeredDataSize / 4。
[out] ScaleToV	将 ADC 采集的原始时域数据转换为电压绝对值（V）的系数。

[out] TriggerInfo	IQ 数据流的触发相关信息。 详见 IQS_TriggerInfo_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef RTA, RTA_Feedback;
RTA_FrameInfo_TypeDef FrameInfo;
RTA_PlotInfo_TypeDef RTA_PlotInfo;
RTA_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
float ScaleToV;
int32_t pointsize = 1;
vector<uint8_t> AlternIQStream(131072);
DataFormat_TypeDef DataFormat = Real16bit;
LookBack_TypeDef LookBackCmd = LookBack_On;
RTA_ProfileDeInit(&Device, &RTA);
RTA.DecimateFactor = 32;
RTA.RBWMode = RBW_Manual;
RTA.RBW_Hz = 1e6;
RTA.SweepTime = 0.0001;
RTA.TriggerAcqTime = 0.1;
if (DataFormat == Real8bit)
    pointsize = 1;
else if (DataFormat == Real16bit)
    pointsize = 2;
Status = RTA_SetDataFormat(&Device, &DataFormat);
Status = RTA_SetLookBackCmd(&Device, &LookBackCmd);
```

```
Status = RTA_Configuration(&Device, &RTA, &RTA_Feedback, &FrameInfo);
vector<uint8_t> SpectrumTrace(FrameInfo.PacketValidPoints *pointsize);
vector<uint16_t> SpectrumBitmap(FrameInfo.FrameHeight *FrameInfo.FrameWidth);
while (1)
{
    RTA_TriggerStart(&Device);
    for (int i = 0; i < FrameInfo.PacketCount; i++) {
        Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace.data(),
        SpectrumBitmap.data(), &RTA_PlotInfo, &TriggerInfo, &MeasAuxInfo);
    }
    for (int i = 0; i < FrameInfo.PacketCount; i++) {
        Status = RTA_GetIQStream(&Device, (void *)AlternIQStream.data(), &ScaleToV,
        &TriggerInfo, &MeasAuxInfo);
        if (Status == APIRETVAL_LastPacket) {
            break;
        }
    }
}
Device_Close(&Device);
```

17.9 RTA_GetRealTimeSpectrum_Raw

```
int RTA_GetRealTimeSpectrum_Raw(
    void** Device,
    uint8_t SpectrumStream[],
    RTA_PlotInfo_TypeDef* PlotInfo,
    RTA_TriggerInfo_TypeDef* TriggerInfo,
    MeasAuxInfo_TypeDef* MeasAuxInfo
)
```

功能描述

获取 RTA 模式下的实时频谱（无概率密度图）及触发相关信息。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。

[out] 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。
[out] RTA_PlotInfo	RTA 获取后返回的绘图信息结构体。 详见 RTA_PlotInfo_TypeDef 结构体的定义。
[out] TriggerInfo	RTA 数据的触发相关信息。 详见 RTA_TriggerInfo_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn, ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
vector<uint8_t> SpectrumTrace(FrameInfo.PacketValidPoints);
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_BusTriggerStart(&Device);
Status = RTA_GetRealTimeSpectrum_Raw(&Device, SpectrumTrace.data(), &PlotInfo, &TriggerInfo, &MeasAuxInfo);
Status = RTA_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

17.10 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

获取实时频谱模式下的实时频谱数据及触发相关的信息。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] 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。
[out] SpectrumBitmap[]	返回概率密度图位图。 数组长度 = FrameInfo.FrameHeight * FrameInfo.FrameWidth。
[out] RTA_PlotInfo	RTA 获取后返回的绘图信息结构体。 详见 RTA_PlotInfo_TypeDef 结构体的定义。
[out] TriggerInfo	RTA 数据的触发相关信息。 详见 RTA_TriggerInfo_TypeDef 结构体的定义。
[out] MeasAuxInfo	返回测量数据的辅助信息。 详见 MeasAuxInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
```

```
RTA_Profile_TypeDef ProfileIn, ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
vector<uint8_t> SpectrumTrace(FrameInfo.PacketValidPoints);
vector<uint16_t> SpectrumBitmap(FrameInfo.FrameHeight * FrameInfo.FrameWidth);
RTA_TriggerInfo_TypeDef TriggerInfo;
RTA_PlotInfo_TypeDef PlotInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = RTA_BusTriggerStart(&Device);
Status = RTA_GetRealTimeSpectrum(&Device, SpectrumTrace.data(), SpectrumBitmap.data(), &PlotInfo, &TriggerInfo, &MeasAuxInfo);
Status = RTA_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

17.11 RTA_SyncTimer

int RTA_SyncTimer(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	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
RTA_Profile_TypeDef ProfileIn, ProfileOut;
RTA_FrameInfo_TypeDef FrameInfo;
```

```
Status = RTA_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.TriggerSource = Timer;
ProfileIn. TriggerTimerSync = SyncToExt_RisingEdge;
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_SyncTimer (&Device);
Status = Device_Close(&Device);
```

18. 列表扫描 MSCAN

18.1 MSCAN_ProfileDeinit

```
int MSCAN_ProfileDeinit(
    void** Device,
    MSCAN_Profile_TypeDef* MSCAN_Profiles,
    int32_t* Elements
)
```

功能描述

初始化配置存储扫描模式的相关参数。

MSCAN 模式下的中心频率、参考电平、驻留时间等参数统一封装在 MSCAN_Profile_TypeDef 结构体中。

兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] MSCAN_Profiles	MSCAN 配置结构体指针。 详见 MSCAN_Profile_TypeDef 结构体的定义。
[out] Elements	MSCAN 配置文件的数量。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 后调用。
示例	请参考 MSCAN_GetData()函数相关示例。

18.2 MSCAN_Configuration

```
int MSCAN_Configuration(
    void** Device,
    MSCAN_Profile_TypeDef* ProfilesIn,
    MSCAN_Profile_TypeDef* ProfilesOut,
    MSCAN_Info_Typedef* MSCAN_Info,
    int32_t* Elements,
    int64_t* Repeatitions,
    PreamplifierState_TypeDef* Preamplifier
)
```

功能描述

配置 MSCAN 的扫描列表和扫描次数等。

兼容性	0.55.79 及之后版本支持

参数说明	
[in] Device	设备句柄。
[in] ProfilesIn	MSCAN 配置结构体指针。 详见 MSCAN_Profile_TypeDef 结构体的定义。
[out] ProfilesOut	MSCAN 配置结构体指针。 详见 MSCAN_Profile_TypeDef 结构体的定义。
[out] MSCAN_Info	返回 MSCAN 频谱帧计数、频谱点数和 IQ 数据点数。 详见 MSCAN_Info_Typedef 结构体的定义。
[out] Elements	MSCAN 配置文件的数量。最大值为 128，超过 128 的值将被重写为 128。
[out] Repeatitions	指向扫描列表中每个 MSCAN 配置文件重复次数的指针。
[out] Preamplifier	预选放大器结构体指针。 详见 PreamplifierState_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 MSCAN_ProfileDeinit 后调用。
示例	请参考 MSCAN_GetData()函数相关示例。

18.3 MSCAN_Start

int MSCAN_Start(void** Device)	
功能描述	
启动列表扫描，按照用户指定的扫描次数(Repeatitions)扫描，扫描完成后自动停止。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 MSCAN_Configuration 之后调用。
示例	请参考 MSCAN_GetData()函数相关示例。

18.4 MSCAN_Stop

int MSCAN_Stop(void** Device)	
功能描述	
终止列表扫描。	
兼容性	0.55.79 及之后版本支持
参数说明	

[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 MSCAN_GetData 之后调用。
示例	请参考 MSCAN_GetData()函数相关示例。

18.5 MSCAN_GetData

```
int MSCAN_GetData(
    void** Device,
    MSCAN_Data_Typedef* MSCAN_Data
)
```

功能描述	
获取扫描的结果。	
兼容性	0.55.79 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] MSCAN_ Data	MSCAN 扫描结果结构体指针。调用 MSCAN_GetData 后，该结构体包含每个扫描配置文件的相关测量数据。 详见 MSCAN_Data_Typedef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 MSCAN_Start 之后调用。
示例	

```
int Status = 0; int DeviceNum = 0; void *Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
int32_t Elements = 95;
int64_t Repeatitions = 100;
std::vector<MSCAN_Profile_TypeDef>  MSCAN_ProfileIn(Elements),  MSCAN_ProfileOut(El
ements);
std::vector<MSCAN_Info_Typedef> MSCAN_Info(Elements);
Status = MSCAN_ProfileDeinit(&Device, MSCAN_ProfileIn.data(), &Elements);
```

```
for (uint32_t i = 0; i < Elements; i++)
{
    MSCAN_ProfileIn[i].CenterFreq_Hz = 50e6 + i * 100e6;
    MSCAN_ProfileIn[i].DwellTime = 16.384e-6;
    MSCAN_ProfileIn[i].DetectCount = 1;
    MSCAN_ProfileIn[i].IQPlayBack = IQPlayBack_Off;
    MSCAN_ProfileIn[i].FFTSize = 2048;
}
PreamplifierState_TypeDef Preamplifier = AutoOn;
Status = MSCAN_Configuration(&Device, MSCAN_ProfileIn.data(), MSCAN_ProfileOut.da
ta(), MSCAN_Info.data(), &Elements, &Repeatitions, &Preamplifier);
// Allocate buffer sized for the largest element.
uint32_t SpectrumFrames_max = 0;
uint32_t SpectrumPoints_max = 0;
uint32_t IQStreamPoints_max = 0;
for (uint32_t i = 0; i < Elements; i++) {
    if (SpectrumFrames_max < MSCAN_Info[i].SpectrumFrames) {
        SpectrumFrames_max = MSCAN_Info[i].SpectrumFrames;
    }
    if (SpectrumPoints_max < MSCAN_Info[i].SpectrumPoints) {
        SpectrumPoints_max = MSCAN_Info[i].SpectrumPoints;
    }
    if (IQStreamPoints_max < MSCAN_Info[i].IQStreamPoints) {
        IQStreamPoints_max = MSCAN_Info[i].IQStreamPoints;
    }
}
std::vector<uint8_t> SpectrumStream(SpectrumFrames_max *SpectrumPoints_max);
std::vector<double> Frequency(SpectrumPoints_max);
std::vector<int16_t> IQStream(IQStreamPoints_max * 2);
MSCAN_Data_Typedef MSCAN_Data;
MSCAN_Data.SpectrumStream = SpectrumStream.data();
MSCAN_Data.IQStream = IQStream.data();
while (1) {
    MSCAN_Start(&Device);
    auto start = std::chrono::high_resolution_clock::now();
```

```
    for (uint32_t i = 0; i < Repeatitions; i++) {
        for (uint32_t t = 0; t < Elements;) {
            Status = MSCAN_GetData(&Device, &MSCAN_Data);
            if (!Status) {
                t = MSCAN_Data.ElementIndex + 1;
            }
            if (MSCAN_Data.ElementIndex == (Elements - 1) && MSCAN_Data.RepeatIndex
             == (Repeatitions - 1)) {
                break;
            }
        }
        if (MSCAN_Data.ElementIndex == (Elements - 1) && MSCAN_Data.RepeatIndex ==
        (Repeatitions - 1)) {
            break;
        }
    }
    auto stop = std::chrono::high_resolution_clock::now();
    auto time = std::chrono::duration_cast<std::chrono::duration<double, std::ratio<1,
    1000>>>(stop - start);
}
Device_Close(&Device);
```

19.数字解调 Digital Demod（选件）

由调制信号频谱图、解调后的星座图、眼图和解调参数组成，深入分析信号的调制质量，提供多项误差指标，有效评估信号在传输中的完整性和可靠性。

19.1 Demod_Check

int Demod_Check()	
功能描述	
核验数字解调库是否存在。	
兼容性	0.55.55 及之后版本支持
返回值	0：解调库存在；-1：解调库不存在。
调用约束	无。
示例	
int Status = -1; Status = Demod_Check();	

19.2 Demod_Open

int Demod_Open(void** Device)	
功能描述	
打开解调功能，检测是否存在许可证和开辟需要的内存。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.3 Demod_Close

int Demod_Close(void** Device)
功能描述
关闭解调功能。

兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Demod_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.4 Demod_Reset

int Demod_Reset(void** Device)	
功能描述	
重置解调功能，当 IQ 数据不连续时，在每次调用 Demod_Execute 前，都需要先调用 Demod_Reset，若 IQ 数据连续，则一直调用 Demod_Execute 即可。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Demod_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.5 Demod_GetVersion

int Demod_Reset(void** Device, char version[])	
功能描述	
获取解调 API 版本。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] version[]	返回解调 API 版本。
返回值	返回 version 的字符长度。
调用约束	在 Demod_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.6 Demod_DeInit

void Demod_DeInit(Demod_Profile_TypeDef* DemodProfile)	
功能描述	
初始化解调配置结构体，将结构体中的每个参数都赋初值。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] DemodProfile	解调配置结构体。 详见 Demod_Profile_TypeDef 结构体的定义。
返回值	无。
调用约束	在 Demod_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.7 Demod_Configuration

int Demod_Configuration(void** Device, const Demod_Profile_TypeDef* DemodProfileIn, Demod_Profile_TypeDef* DemodProfileOut)	
功能描述	
配置解调参数。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] DemodProfileIn	输入解调配置结构体。 详见 Demod_Profile_TypeDef 结构体的定义。
[out] DemodProfileOut	输出解调配置结构体，若输入配置不合理，将回写输出配置结构体为合理参数。 详见 Demod_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Demod_Open 之后进行调用。
示例	请参考 Demod_Execute()函数相关示例。

19.8 Demod_Execute

```
int Demod_Execute(
    void** Device,
    const IQStream_TypeDef* IQStream,
    DemodInfo_TypeDef* DemodInfo
)
```

功能描述	
配置解调参数。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] IQStream	IQ 数据流，包括 IQ 数据及相关配置信息等。 详见 IQStream_TypeDef 结构体的定义。
[out] DemodInfo	解调信息结构体，包括：眼图、星座图、EVM 等，注意：结构体中所有指针变量将指向函数内部的空间，无需用户在外部手动开辟。 详见 DemodInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Demod_Open 之后进行调用。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef IQS_ProfileIn, IQS_ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
IQS_TriggerInfo_TypeDef TriggerInfo;
IQS_ProfileDeInit(&Device, &IQS_ProfileIn);
IQS_ProfileIn.CenterFreq_Hz = 1e9;
IQS_ProfileIn.RefLevel_dBm = 0;
IQS_ProfileIn.DataFormat = Complex16bit; //注意：解调函数目前只能输入 int16 类型的 IQ 数据
IQS_ProfileIn.TriggerMode = FixedPoints;
IQS_ProfileIn.TriggerSource = Bus;
```

```
IQS_ProfileIn.DecimateFactor = 4;
IQS_ProfileIn.TriggerLength = 32484;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
vector<int16_t> IQ(StreamInfo.StreamSamples * 2);
vector<int16_t> I(StreamInfo.StreamSamples);
vector<int16_t> Q(StreamInfo.StreamSamples);
Status = Demod_Open(&Device);
vector<char> version(50);
Demod_GetVersion(&Device, version.data());
Demod_Profile_TypeDef DemodProfileIn, DemodProfileOut;
DemodInfo_TypeDef DemodInfo;
Demod_DeInit(&DemodProfileIn);
DemodProfileIn.SamplePoints = StreamInfo.StreamSamples;
DemodProfileIn.SampleRate = StreamInfo.IQSampleRate;
DemodProfileIn.ModType = QAM16;
DemodProfileIn.SymbolRate = 1e6;
Status = Demod_Configuration(&Device, &DemodProfileIn, &DemodProfileOut);
IQS_ProfileIn.TriggerLength = DemodProfileOut.SamplePoints;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
DemodProfileIn.SamplePoints = StreamInfo.StreamSamples;
Status = Demod_Configuration(&Device, &DemodProfileIn, &DemodProfileOut);
while (1) {
  uint32_t Points = StreamInfo.PacketSamples;
  Status = IQS_BusTriggerStart(&Device);
  for (uint32_t i = 0; i < StreamInfo.PacketCount; i++) {
    Status = IQS_GetIQStream_PM1(&Device, &IQStream);
    if (i == StreamInfo.PacketCount - 1 && StreamInfo.StreamSamples % StreamInfo.P
    acketSamples != 0) {
      Points = StreamInfo.StreamSamples % StreamInfo.PacketSamples;
    }
      memcpy(IQ.data() + i * StreamInfo.PacketSamples * 2, IQStream.AlternIQStream,
      Points * 2 * sizeof(IQ[0]));
  }
  IQStream.AlternIQStream = IQ.data();
```

```
    //若每次做解调的 IQ 数据不是连续的，则需要先调用 Demod_Reset，再调用 Demod_Execute
    Demod_Reset(&Device);
    Status = Demod_Execute(&Device, &IQStream, &DemodInfo);
}
Status = Demod_Close(&Device);
Status = Device_Close(&Device);
```

19.9 Demod_GenSymbolMap

```
void Demod_GenSymbolMap(
    Demod_ModType_TypeDef ModType,
    Demod_SymbolMap_TypeDef SymbolMap[1024],
    uint32_t* MapNum
)
```

功能描述

根据输入的调制类型，生成一个符号映射表，并计算出符号映射表中符号的数量。

兼容性	0.55.55 及之后版本支持
参数说明	
[in] ModType	调制类型，可解调 2ASK、2FSK、4FSK、GMSK、BPSK、QPSK、8PSK、16QAM、32QAM、64QAM、128QAM 和 256QAM 信号。
[out] SymbolMap[1024]	表示符号映射表中的单个符号。该符号的映射表是用于调制解调过程中的符号与坐标之间的关系映射。 详见 Demod_SymbolMap_TypeDef 结构体的定义。
[out] MapNum	符号映射表中可用的符号个数。
返回值	无。
调用约束	无。

示例

```
int Status = -1;
Demod_ModType_TypeDef ModType = QPSK;
Demod_SymbolMap_TypeDef SymbolMap[1024];
uint32_t MapNum = 0;
Demod_GenSymbolMap(ModType, SymbolMap, &MapNum);
```

20. 脉冲检测 Pulse Det（选件）

20.1 Pulse_Open

int Pulse_Open(void** Device)	
功能描述	
打开脉冲检测功能，检查是否存在所需许可证，并在内存中分配运行脉冲检测所需的空间。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄，函数调用后将返回脉冲检测功能所使用的内存地址，用于后续脉冲检测函数的索引。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 之后进行调用，必须在调用其它脉冲检测相关函数之前调用。
示例	请参考 Pulse_Detect()函数相关示例。

20.2 Pulse_Close

int Pulse_Close(void** Device)	
功能描述	
关闭脉冲检测功能，释放之前 Pulse_Open 分配的内存空间。在整个脉冲检测操作完成后调用，释放资源。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄，指向 Pulse_Open 返回的脉冲检测功能内存地址。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 和 Pulse_Open 之后进行调用。
示例	请参考 Pulse_Detect()函数相关示例。

20.3 Pulse_Detect

```
int Pulse_Detect(
    void** Device,
    const Pulse_Profile_TypeDef* Pulse_Profile,
    PulseInfo_TypeDef* PulseInfo
)
```

功能描述	
对 DET 数据执行脉冲检测功能，基于输入的脉冲数据和阈值参数，分析并输出脉冲特性信息，如脉宽、周期和占空比等。必须在 Pulse_Open 成功后调用。	
兼容性	0.55.55 及之后版本支持
参数说明	
[in] **Device**	设备句柄。
[in] **Pulse_Profile**	输入脉冲检测数据，包括：需要检测的数据、阈值等。 详见 Pulse_Profile_TypeDef 结构体的定义。
[out] **PulseInfo**	输出脉冲检测结果，包括：脉宽、周期、占空比等。 详见 PulseInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 和 Pulse_Open 之后进行调用。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
Status = Pulse_Open (&Device);
DET_Profile_TypeDef DET_ProfileIn, DET_ProfileOut;
DET_StreamInfo_TypeDef StreamInfo;
DET_ProfileDeInit(&Device, &DET_ProfileIn);
DET_ProfileIn.CenterFreq_Hz = 1e9;
DET_ProfileIn.RefLevel_dBm = 0;
DET_ProfileIn.DecimateFactor = 2;
DET_ProfileIn.DecimateFactor = 1;
DET_ProfileIn.TriggerMode = FixedPoints;
DET_ProfileIn.TriggerSource = Bus;
DET_ProfileIn.TriggerLength = 16242 * 10;
Status = DET_Configuration(&Device, &DET_ProfileIn, &DET_ProfileOut, &StreamInfo);
vector<float>                NormalizedPowerStream(StreamInfo.PacketCount                *
StreamInfo.PacketSamples);
```

```
float ScaleToV = 0;
DET_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = DET_BusTriggerStart(&Device);
for (uint32_t i = 0; i < StreamInfo.PacketCount; i++) {
  Status = DET_GetPowerStream(&Device, NormalizedPowerStream.data() + i *
  StreamInfo.PacketSamples, &ScaleToV, &TriggerInfo, &MeasAuxInfo);
}
//将 DET 数据的单位转换成 dBm
vector<float> NormalizedPowerStream_dBm(NormalizedPowerStream.size());
for (int i = 0; i < StreamInfo.StreamSamples; i++) {
  NormalizedPowerStream_dBm[i] = 10 * log10(20 * pow(NormalizedPowerStream[i] *
  ScaleToV, 2));
}
Status = Pulse_Open(&Device); //打开脉冲检测功能
Pulse_Profile_TypeDef Pulse_Profile;
Pulse_Profile.TimeResolution_s = StreamInfo.TimeResolution;
Pulse_Profile.PulseSize = NormalizedPowerStream_dBm.size();
Pulse_Profile.unit = Power_dBm;
Pulse_Profile.DetThreshold = -20;
Pulse_Profile.ExpPulseNum = 10;
Pulse_Profile.Pulse = NormalizedPowerStream_dBm.data();
PulseInfo_TypeDef PulseInfo;
Status = Pulse_Detect(&Device, &Pulse_Profile, &PulseInfo);
Status = Pulse_Close(&Device);
Status = Device_Close(&Device);
```

20.4 Pulse_Detect_PM1

```
int Pulse_Detect_PM1(
    void** Device,
    const Pulse_Profile_TypeDef* Pulse_Profile,
    PulseInfoPM1_TypeDef* PulseInfoPM1
)
```

功能描述

对 IQ 数据执行脉冲检测功能，基于输入的脉冲数据和阈值参数，分析并输出脉冲特性信息，如脉宽、周期、占空比、脉冲调制类型、脉冲的频率和相位信息等。必须在 Pulse_Open 成功后调用。

兼容性	0.55.55 及之后版本支持
参数说明	
[in] Device	设备句柄。
[in] Pulse_Profile	输入脉冲检测数据，包括：需要检测的数据、阈值等。 详见 Pulse_Profile_TypeDef 结构体的定义。
[out] PulseInfoPM1	输出脉冲检测结果，包括脉冲调制类型、脉冲的频率和相位信息等。 详见 PulseInfoPM1_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 Device_Open 和 Pulse_Open 之后进行调用。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
Status = Pulse_Open(&Device);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.CenterFreq_Hz = 1.00e9;
ProfileIn.DecimateFactor = 2;
ProfileIn.TriggerLength = 16242*100;
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
```

```
void* AlternIQStream = NULL;
float ScaleToV = 0;
vector<float> IQ_Data(StreamInfo.StreamSamples * 2);
IQS_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = IQS_BusTriggerStart(&Device);
for (int j = 0; j < StreamInfo.PacketCount; j++) {
    Status = IQS_GetIQStream(&Device, &AlternIQStream, &ScaleToV, &TriggerInfo, &MeasAuxInfo);
    int16_t* IQ = (int16_t*)AlternIQStream;
    for (int i = 0; i < StreamInfo.PacketSamples * 2; i ++){
        IQ_Data[i + StreamInfo.PacketSamples * 2 * j] = (float)IQ[i] * ScaleToV;
    }
}
Status = IQS_BusTriggerStop(&Device);
Pulse_Profile_TypeDef Pulse_Profile;
Pulse_Profile.TimeResolution_s = 1.0 / StreamInfo.IQSampleRate;
Pulse_Profile.PulseSize = IQ_Data.size() / 2;
Pulse_Profile.unit = Power_dBm;
Pulse_Profile.DetThreshold = -20;
Pulse_Profile.ExpPulseNum = 10;
Pulse_Profile.Pulse = IQ_Data.data();
PulseInfoPM1_TypeDef PulseInfoPM1;
Status = Pulse_Detect_PM1(&Device, &Pulse_Profile, &PulseInfoPM1);
Status = Pulse_Close(&Device);
Status = Device_Close(&Device);
```

21. 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。

21.1 ASG_ProfileDeInit

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

功能描述	
将辅助信号源的配置参数初始化为默认状态，用于清空或重置 Profile 中的各项设置，确保后续配置从标准初始状态开始，避免受历史配置影响。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Device	设备句柄。
[out] Profile	ASG 配置结构体指针。 详见 ASG_Profile_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要 Device_Open 后调用。
示例	请参考 ASG_Configuration()函数相关示例。

21.2 ASG_Configuration

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

功能描述	
配置模拟信号源的工作状态，根据输入的 Profile 参数设置信号源的工作模式、频率、功率、扫描及触发等配置，并返回实际生效的配置信息及当前信号源状态信息。	
兼容性	0.55.77 及之后版本支持
参数说明	

[in] **Device**	设备句柄引用，用于索引当前已打开的设备实例。
[in] **ProfileIn**	输入的模拟信号源配置结构体，用于设置信号源的工作参数。 详见 ASG_Profile_TypeDef 结构体的定义。
[out] **ProfileOut**	输出的模拟信号源实际生效的配置参数，反映当前设备中已应用的 ASG 配置。 详见 ASG_Profile_TypeDef 结构体的定义。
[out] **ASG_Info**	ASG 模式下，ASG 相关信息。 详见 ASG_Info_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 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;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
ASG_Profile_TypeDef ProfileIn, ProfileOut;
ASG_Info_TypeDef ASG_Info;
Status = ASG_ProfileDeInit(&Device, &ProfileIn);
ProfileIn.CenterFreq_Hz = 1e9;
Status = ASG_Configuration(&Device, &ProfileIn, &ProfileOut, &ASG_Info);
Status = Device_Close(&Device);
```

22. 模拟信号解调 ADM

ADM 为模拟调制解调处理接口，用户可调用其中的函数做模拟信号解调处理。

22.1 ADM_Open

void ADM_Open(void** ADM)	
功能描述	
创建并初始化模拟解调功能实例，在内存中分配模拟解调所需的资源。在调用任何模拟解调相关函数之前，必须先调用该函数。	
兼容性	0.55.77 及之后版本支持
参数说明	
[out] ADM	模拟解调功能的内存引用。调用成功后返回已创建的 ADM 实例地址，后续所有模拟解调相关 API 均需通过该指针进行操作。
返回值	无。
调用约束	在其它模拟解调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 ADM_Open 调用，必须在整个功能使用需求结束之后，调用 ADM_Close 函数，以释放内存。
示例	请参考 ADM_AMDemod_PM1()函数相关示例。

22.2 ADM_Close

void ADM_Close(void** ADM)	
功能描述	
关闭并释放模拟解调功能实例，释放 ADM_Open 分配的内部资源与内存空间。在模拟解调功能不再使用时必须调用该函数。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] ADM	模拟解调功能的内存引用，由 ADM_Open 返回。调用后该实例失效，其内部资源被释放。
返回值	无。

调用约束	只需在程序执行的最后调用此函数，调用此函数后模拟解调功能关闭，内存空间释放。如需要重新调用模拟解调功能，则再次通过调用 ADM_Open，打开 Analog 功能。
示例	请参考 ADM_AMDemod_PM1()函数相关示例。

22.3 ADM_AMDemod

```
int ADM_AMDemod(
    void** ADM,
    const void* AlternIQStream,
    DataFormat_TypeDef DataFormat,
    uint64_t SamplePoints,
    double IQSampleRate,
    float DemodWaveform[]
)
```

功能描述	
对输入的 IQ 数据执行 AM 解调，仅输出解调后的时域波形数据。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] ADM	模拟解调功能实例引用，由 ADM_Open 创建，用于索引当前 ADM 解调上下文。
[in] AlternIQStream	输入的交错 IQ 数据缓冲区，数据格式由 DataFormat 指定。
[in] DataFormat	输入 IQ 数据的数据格式，用于指示 IQ 数据在内存中的排列与精度。详见 DataFormat_TypeDef 枚举的定义。
[in] SamplePoints	输入 IQ 数据的采样点数。
[in] IQSampleRate	输入 IQ 数据的采样率，单位：Hz。
[out] DemodWaveform	AM 解调后的时域波形数据数组，长度与 SamplePoints 对应。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后模拟解调功能关闭，内存空间释放。如需要重新调用模拟解调功能，则再次通过调用 ADM_Open，打开 Analog 功能。
示例	请参考 ADM_AMDemod_PM1()函数相关示例。

22.4 ADM_FMDemod

```
int ADM_FMDemod(
    void** ADM,
    const void* AlternIQStream,
    DataFormat_TypeDef DataFormat,
    uint64_t SamplePoints,
    double IQSampleRate,
    bool reset,
    float DemodWaveform[]
)
```

功能描述	
对输入的 IQ 数据执行 FM 解调，仅输出解调后的时域波形数据。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] ADM	模拟解调功能实例引用，由 ADM_Open 创建，用于索引当前 ADM 解调上下文。
[in] AlternIQStream	输入的交错 IQ 数据缓冲区，数据格式由 DataFormat 指定。
[in] DataFormat	输入 IQ 数据的数据格式，用于指示 IQ 数据在内存中的排列与精度。 详见 DataFormat_TypeDef 枚举的定义。
[in] SamplePoints	输入 IQ 数据的采样点数。
[in] IQSampleRate	输入 IQ 数据的采样率，单位：Hz。
[in] reset	是否重置解调器状态。 true 表示在解调前清除上次残留状态，false 表示保持累积状态。
[out] DemodWaveform	FM 解调后的时域波形数据数组，长度与 SamplePoints 对应。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 ADM_Open 之后进行调用。
示例	请参考 ADM_FMDemod_PM1()函数相关示例。

22.5 ADM_AMDemod_PM1

```
int ADM_AMDemod_PM1(
    void** ADM,
    const void* AlternIQStream,
    DataFormat_TypeDef DataFormat,
    uint64_t SamplePoints,
    double IQSampleRate,
    float IQS_ScaleToV,
    AMDemodParam_TypeDef* AMDemodParam
)
```

功能描述	
对输入的 IQ 数据执行 AM 解调处理，在输出解调后波形的同时，计算并返回幅度调制相关参数，用于 AM 信号的调制特性分析。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] ADM	模拟解调功能实例引用，由 ADM_Open 创建，用于索引当前 ADM 解调上下文。
[in] AlternIQStream	返回当前设备的 GNSS 卫星信息，包括可视卫星数量和用于定位的卫星信息。
[in] DataFormat	输入 IQ 数据的数据格式，用于指示 IQ 数据在内存中的排列与精度。 详见 DataFormat_TypeDef 枚举的定义。
[in] SamplePoints	输入 IQ 数据的采样点数。
[in] IQSampleRate	输入 IQ 数据的采样率，单位：Hz。
[in] IQS_ScaleToV	IQ 采样值转换为电压的比例系数，用于 AM 调制参数的物理量计算。
[out] AMDemodParam	返回 AM 解调后的调制参数及相关分析结果。 详见 AMDemodParam_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 ADM_Open 之后进行调用。
示例	

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

```
Device_Open_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo);
IQStream_TypeDef IQStream;
IQS_Profile_TypeDef IQS_ProfileIn, IQS_ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &IQS_ProfileIn);
IQS_ProfileIn.CenterFreq_Hz = 1e9;
IQS_ProfileIn.RefLevel_dBm = 0;
IQS_ProfileIn.DataFormat = Complex16bit;
IQS_ProfileIn.TriggerMode = FixedPoints;
IQS_ProfileIn.TriggerSource = Bus;
IQS_ProfileIn.DecimateFactor = 256;
IQS_ProfileIn.BusTimeout_ms = 5000;
IQS_ProfileIn.TriggerLength = 16242 * 1;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
IQS_Configuration_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo, &I
QS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
AMDemodParam_TypeDef AMDemodParam; //Demod
void* ADM = nullptr;
ADM_Open(&ADM);
vector<int16_t> IQ(StreamInfo.StreamSamples * 2);
vector<float> DemodWaveform(IQS_ProfileIn.TriggerLength);
while(1) {
  uint32_t Points = StreamInfo.PacketSamples;
  Status = IQS_BusTriggerStart(&Device);
  for (uint32_t i = 0; i < StreamInfo.PacketCount; i++) {
    Status = IQS_GetIQStream_PM1(&Device, &IQStream);
     if (i == StreamInfo.PacketCount - 1 && StreamInfo.StreamSamples % StreamInfo.P
   acketSamples != 0) {
        Points = StreamInfo.StreamSamples % StreamInfo.PacketSamples;
    }
    memcpy(IQ.data() + i * StreamInfo.PacketSamples * 2, IQStream.AlternIQStream, P
    oints * 2 * sizeof(IQ[0]));
  }
  IQStream.AlternIQStream = IQ.data();
  Status = ADM_AMDemod(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.Data
  Format, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate,
  DemodWaveform.data());
  Status = ADM_AMDemod_PM1(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profil
  e.DataFormat, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampl
  eRate, IQStream.IQS_ScaleToV, &AMDemodParam);
}
ADM_Close(&ADM);
Status = Device_Close(&Device);
```

22.6 ADM_FMDemod_PM1

```
int ADM_FMDemod_PM1(
    void** ADM,
    const void* AlternIQStream,
    DataFormat_TypeDef DataFormat,
    uint64_t SamplePoints,
    double IQSampleRate,
    float IQS_ScaleToV,
    bool reset,
    FMDemodParam_TypeDef* FMDemodParam
)
```

功能描述

对输入的 IQ 数据执行 FM 解调处理，同时返回解调后的波形和调制参数，用于分析 FM 信号的调制特性。可选择在解调前重置解调器状态。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] ADM	模拟解调功能实例引用，由 ADM_Open 创建，用于索引当前 ADM 解调上下文。
[in] AlternIQStream	输入的交错 IQ 数据缓冲区，数据格式由 DataFormat 指定。
[in] DataFormat	输入 IQ 数据的数据格式，用于指示 IQ 数据在内存中的排列与精度。详见 DataFormat_TypeDef 枚举的定义。
[in] SamplePoints	输入 IQ 数据的采样点数。
[in] IQSampleRate	输入 IQ 数据的采样率，单位：Hz。
[in] IQS_ScaleToV	IQ 采样值转换为电压的比例系数，用于 AM 调制参数的物理量计算。
[in] 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。
[out] FMDemodParam	返回 FM 解调后的调制参数及相关分析结果。详见 FMDemodParam_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 ADM_Open 之后进行调用。

示例

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef  BootProfile;
BootInfo_TypeDef  BootInfo;
BootProfile.DevicePowerSupply  =  USBPortAndPowerPort;
```

```
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
IQStream_TypeDef IQStream;
IQS_Profile_TypeDef IQS_ProfileIn;
IQS_Profile_TypeDef IQS_ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &IQS_ProfileIn);
IQS_ProfileIn.CenterFreq_Hz = 1e9;
IQS_ProfileIn.RefLevel_dBm = 0;
IQS_ProfileIn.DataFormat = Complex16bit;
IQS_ProfileIn.TriggerMode = FixedPoints;
IQS_ProfileIn.TriggerSource = Bus;
IQS_ProfileIn.DecimateFactor = 256;
IQS_ProfileIn.BusTimeout_ms = 5000;
IQS_ProfileIn.TriggerLength = 16242 * 1;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
FMDemodParam_TypeDef FMDemodParam; //Demod
void* ADM = nullptr;
ADM_Open(&ADM);
vector<int16_t> IQ(StreamInfo.StreamSamples * 2);
vector<float> DemodWaveform(IQS_ProfileIn.TriggerLength);
while(1) {
  uint32_t Points = StreamInfo.PacketSamples;
  Status = IQS_BusTriggerStart(&Device);
  for (uint32_t i = 0; i < StreamInfo.PacketCount; i++) {
    Status = IQS_GetIQStream_PM1(&Device, &IQStream);
    if (i == StreamInfo.PacketCount - 1 && StreamInfo.StreamSamples % StreamInfo.P
      acketSamples != 0) {
        Points = StreamInfo.StreamSamples % StreamInfo.PacketSamples;
    }
      memcpy(IQ.data() + i * StreamInfo.PacketSamples * 2, IQStream.AlternIQStream,
      Points * 2 * sizeof(IQ[0]));
  }
  IQStream.AlternIQStream = IQ.data();
  Status = ADM_FMDemod(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.Data
  Format, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate,
  true,DemodWaveform.data());
  Status = ADM_FMDemod_PM1(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.
  DataFormat, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleR
  ate, IQStream.IQS_ScaleToV, true, &FMDemodParam);
}
ADM_Close(&ADM);
Status = Device_Close(&Device);
```

23. 数字信号处理 DSP 迹线分析

23.1 DSP_TraceAnalysis_IM3

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

功能描述

对输入的频谱迹线数据进行 IM3 参数分析，基于频率与功率谱数据计算并输出三阶互调相关结果。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[out] IM3Result	返回三阶互调分析结果，包括主信号频率、功率、互调产物及 IP3 计算结果。 详见 TraceAnalysisResult_IP3_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Get 之后进行调用。

示例

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
```

```
uint8_t IfDoConfig = 1;
Status = SWP_AutoSet(&Device, SWPIM3Meas, &SWP_ProfileIn, &ProfileSetOut, &TraceInfo, IfDoConfig);
ProfileSetOut.Span_Hz = 10e6;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
TraceAnalysisResult_IP3_TypeDef  IM3Result;
Status = DSP_TraceAnalysis_IM3(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, &IM3Result);
Status = Device_Close(&Device);
```

23.2 DSP_TraceAnalysis_IM2

```
int DSP_TraceAnalysis_ IM2(
    const double Freq_Hz[],
    const float PowerSpec_dBm[],
    uint32_t TracePoints,
    TraceAnalysisResult_IP2_TypeDef* IM2Result
)
```

功能描述

对输入的频谱迹线数据进行 IM2 参数分析，基于频率与功率谱数据计算并输出二阶互调相关结果。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[out] IM2Result	返回二阶互调分析结果，包括主信号频率、功率、二阶互调产物及 IP2 计算结果。 详见 TraceAnalysisResult_IP2_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。

调用约束	需要在 SWP_Get 之后进行调用。
示例	

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
uint8_t IfDoConfig = 1;
Status = SWP_AutoSet(&Device, SWPIM3Meas, &SWP_ProfileIn, &ProfileSetOut, &TraceInfo,
IfDoConfig);
ProfileSetOut.Span_Hz = 10e6;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &Meas
AuxInfo);
TraceAnalysisResult_IP2_TypeDef IM2Result;
Status = DSP_TraceAnalysis_IM2(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.Fu
llsweepTracePoints, &IM2Result);
Status = Device_Close(&Device);
```

23.3 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	
对输入的频谱迹线数据进行信道功率分析，根据中心频率、分析带宽和分辨带宽计算指定信道内的功率，并输出分析结果。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[in] CenterFrequency	需要测量的信道中心频率。
[in] AnalysisSpan	需要测量的信道带宽。
[in] RBW	分析使用的分辨率带宽。
[out] ChannelPowerResult	返回信道功率分析结果，包括信道总功率、功率密度以及信道内峰值的频率、功率和索引信息。 详见 DSP_ChannelPowerInfo_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Get 之后进行调用。
示例	

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

```
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
uint8_t IfDoConfig = 1;
SWP_AutoSet(&Device, SWPChannelPowerMeas, &SWP_ProfileIn, &ProfileSetOut, &TraceInfo, IfDoConfig);
ProfileSetOut.Span_Hz = 10e6;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
double CenterFrequency = 1e9;
double AnalysisSpan = 2e6;
DSP_ChannelPowerInfo_TypeDef ChannelPowerResult;
Status = DSP_TraceAnalysis_ChannelPower(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, CenterFrequency, AnalysisSpan, SWP_ProfileOut.RBW_Hz, &ChannelPowerResult);
Status = Device_Close(&Device);
```

23.4 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	
对输入的频谱迹线数据进行 XdB 带宽分析，基于功率谱计算相对于峰值下降 XdB 时对应的信号带宽，并输出分析结果。	
兼容性	0.55.77 及之后版本支持

参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[in] XdB	相对于峰值功率的下降量，用于定义 XdB 带宽。
[out] XdBResult	返回 XdB 带宽分析结果，包括带宽大小、中心频率、起止频率，以及带宽内峰值的频率、功率和索引信息。 详见 TraceAnalysisResult_XdB_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Get 之后进行调用。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
uint8_t IfDoConfig = 1;
Status = SWP_AutoSet(&Device, SWPOBWMeas, &SWP_ProfileIn, &ProfileSetOut, &TraceInfo, IfDoConfig);
ProfileSetOut.RBW_Hz = 50e3;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
float XdB = 3;
```

```
TraceAnalysisResult_XdB_TypeDef XdBResult;
Status = DSP_TraceAnalysis_XdBBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, XdB, &XdBResult);
Status = Device_Close(&Device);
```

23.5 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

对输入的频谱迹线数据进行占用带宽分析，计算给定功率占比下的信号带宽，并输出分析结果。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[in] OccupiedRatio	需要测试的占用带宽比例，通常设置为 0.99。
[out] OBWResult	返回占用带宽分析结果，包括占用带宽大小、中心频率、起止频率及对应功率、功率占比，以及带宽内峰值的频率、功率和索引信息。 详见 TraceAnalysisResult_OBW_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Get 之后进行调用。

示例

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

```
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
uint8_t IfDoConfig = 1;
Status = SWP_AutoSet(&Device, SWPOBWMeas, &SWP_ProfileIn, &ProfileSetOut, &TraceInfo, IfDoConfig);
ProfileSetOut.Span_Hz = 10e6;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);

vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
float OccupiedRatio = 0.99;
TraceAnalysisResult_OBW_TypeDef OBWResult;
Status = DSP_TraceAnalysis_OBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, OccupiedRatio, &OBWResult);
Status = Device_Close(&Device);
```

23.6 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	
对输入的频谱迹线数据进行邻道功率比分析，根据指定邻道频率信息计算主信道与邻道的功率比，并输出分析结果。	
兼容性	0.55.77 及之后版本支持

参数说明	
[in] Freq_Hz[]	输入的频率数组，对应迹线中各点的频率值，单位 Hz。
[in] PowerSpec_dBm[]	输入的功率数组，对应迹线中各点的功率值，单位 dBm。
[in] TracePoints	迹线点数，表示频率数组与功率数组的长度。
[in] ACPRFreqInfo	输入邻道功率比分析所需的频率信息，包括分辨率带宽、主信道中心频率与带宽、邻道间隔以及邻道对数量。 详见 DSP_ACPRFreqInfo_TypeDef 结构体的定义。
[out] ACPRResult	返回邻道功率比分析结果，包括主信道功率及峰值信息，左右邻道的中心频率、带宽、功率、功率比、功率差及峰值信息。 详见 TraceAnalysisResult_ACPR_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 SWP_Get 之后进行调用。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, ProfileSetOut, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.FreqAssignment = CenterSpan;
uint8_t IfDoConfig = 1;
Status = SWP_AutoSet(&Device, SWPACPRMeas, &SWP_ProfileIn, &ProfileSetOut,
&TraceInfo, IfDoConfig);
ProfileSetOut.Span_Hz = 10e6;
Status = SWP_Configuration(&Device, &ProfileSetOut, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(),
&MeasAuxInfo);
```

```
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo;
ACPRFreqInfo.RBW = SWP_ProfileOut.RBW_Hz;
ACPRFreqInfo.AdjChPair = 2;
ACPRFreqInfo.AdjChSpace_Hz = 2e6;
ACPRFreqInfo.MainChBW_Hz = 1e6;
ACPRFreqInfo.MainChCenterFreq_Hz = 1e9;
vector<TraceAnalysisResult_ACPR_TypeDef> ACPRResult(ACPRFreqInfo.AdjChPair);
Status = DSP_TraceAnalysis_ACPR(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.
FullsweepTracePoints, ACPRFreqInfo, ACPRResult.data());
Status = Device_Close(&Device);
```

24. 数字信号处理 DSP 流数据的分析与处理

24.1 DSP_Open

int DSP_Open(void** DSP)	
功能描述	
打开 DSP 功能，并在内存中分配存储 DSP 相关数据的空间。在调用其他 DSP 函数之前必须先调用此函数。可通过提供多个 DSP 指针同时操作多个 DSP 实例。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	运行 DSP 所需的内存空间引用。调用后返回当前打开的 DSP 功能内存地址，后续调用其他 API 时需使用该引用索引此地址。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	必须在所有其他 DSP 函数调用之前调用，仅需在程序开始时调用一次。使用结束后，必须调用 DSP_Close 释放内存。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.2 DSP_Close

int DSP_ Close(void** DSP)	
功能描述	
关闭 DSP 功能，释放之前由 DSP_Open 分配的内存空间。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	运行 DSP 所需的内存空间引用，指向由 DSP_Open 返回的内存地址，用于后续释放或操作 DSP 功能。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	必须在程序执行结束时调用此函数。调用后 DSP 功能关闭并释放内存。如需再次使用 DSP 功能，必须重新调用 DSP_Open。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.3 DSP_FFT_DeInit

int DSP_FFT_DeInit(DSP_FFT_TypeDef* IQToSpectrum)	
功能描述	
初始化 FFT 模式相关参数，包括 FFT 点数、窗函数类型、抽取倍数等，所有参数封装在 DSP_FFT_TypeDef 结构体中。	
兼容性	0.55.77 及之后版本支持
参数说明	
[out] IQToSpectrum	配置 FFT 模式的参数结构体，包括分析点数、有效采样点数、窗型、迹线检波方式、检测比、频谱截取比例及是否进行校准。 详见 DSP_FFT_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 DSP_FFT_Configuration 前调用。
示例	请参考 DSP_FFT_IQSToSpectrum()函数相关示例。

24.4 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	
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	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] ProfileIn	输入 FFT 配置参数，包括分析点数、采样点数、窗型、迹线检波方式等。 详见 DSP_FFT_TypeDef 结构体的定义。
[out] ProfileOut	返回实际应用的 FFT 配置参数，可用于确认配置。 详见 DSP_FFT_TypeDef 结构体的定义。

[out] TracePoints	当前 DSP_FFT 配置下可获取的频谱点数。
[out] RBWRatio	返回分辨率带宽相对于采样率的比值。 计算公式：RBW = RBWRatio * IQSampleRate
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DSP_FFT_DeInit 之后进行调用。
示例	请参考 DSP_FFT_IQSToSpectrum()函数相关示例。

24.5 DSP_FFT_IQSToSpectrum

```
int DSP_FFT_IQSToSpectrum(
    void** DSP,
    const IQStream_TypeDef* IQStream,
    double Freq_Hz[],
    float PowerSpec_dBm[]
)
```

功能描述	
将输入的 IQ 数据流转换为频谱数据，输出对应的频率和功率数组，便于进行频谱分析。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] IQStream	IQ 数据流的相关信息，包括 IQ 数据及相关配置信息。 详见 IQStream_TypeDef 结构体的定义。
[out] Freq_Hz[]	返回频谱数据的频率数组，单位 Hz，数组长度为 TracePoints，由 DSP_FFT_Configuration()函数确定。
[out] PowerSpec_dBm[]	返回频谱数据的功率数组，单位 dBm，数组长度为 TracePoints，由 DSP_FFT_Configuration()函数确定。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DSP_FFT_Configuration 之后进行调用。
示例	

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

```
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
IQStream_TypeDef IQStream;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
vector<int16_t> AlternIQStream(StreamInfo.StreamSamples * 2);
void* DSP = NULL; uint32_t TracePoints = 0; double RBWRatio = 0;
Status = DSP_Open(&DSP);
DSP_FFT_TypeDef FFT_ProfileIn, FFT_ProfileOut;
Status = DSP_FFT_DeInit(&FFT_ProfileIn);
Status = DSP_FFT_Configuration(&DSP, &FFT_ProfileIn, &FFT_ProfileOut, &TracePoints, &RBWRatio);
vector<double> Frequency(TracePoints); vector<float> PowerSpec_dBm(TracePoints);
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = DSP_FFT_IQSToSpectrum(&DSP, &IQStream, Frequency.data(), PowerSpec_dBm.data());
Status = IQS_BusTriggerStop(&Device);
Status = DSP_Close(&DSP);
Status = Device_Close(&Device);
```

24.6 DSP_DDC_DeInit

int DSP_DDC_DeInit(DSP_DDC_TypeDef* DDC_ProfileIn)	
功能描述	
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	0.55.77 及之后版本支持
参数说明	
[out] DDC_ProfileIn	输入的数字下变频（DDC）配置参数，包括复混频偏移频率、采样率、重采样抽取倍数以及采样点数，用于初始化或配置 DDC 功能。 详见 DSP_DDC_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	在 DSP_DDC_Configuration 前调用。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.7 DSP_DDC_Configuration

int DSP_DDC_Configuration(void** DSP, const DSP_DDC_TypeDef* DDC_ProfileIn, DSP_DDC_TypeDef* DDC_ProfileOut)	
功能描述	
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	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] ProfileIn	输入的 DDC 配置参数，包括复混频频率偏移、采样率、抽取倍数和采样点数，用于设置 DDC 功能。 详见 DSP_DDC_TypeDef 结构体的定义。
[out] ProfileOut	返回实际应用的 DDC 配置参数，可用于确认配置。 详见 DSP_DDC_TypeDef 结构体的定义。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DSP_DDC_DeInit 之后进行调用。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.8 DSP_DDC_Reset

void DSP_DDC_Reset(void** DSP)	
功能描述	
重置 DDC 功能中的缓存，清除历史数据以准备新的下变频处理。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in/out] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
返回值	无。
调用约束	需要在 DSP_Open 之后进行调用。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.9 DSP_DDC_GetDelay

```
void DSP_DDC_GetDelay(
        void** DSP,
        uint32_t* delay
)
```

功能描述	
获取当前 DDC 的处理延时。通过提供 DSP 内存引用，返回 DDC 在当前配置下的延时（单位为采样点数），可用于校准或同步处理。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[out] delay	返回 DDC 的处理延时，单位为采样点数。
返回值	无。
调用约束	需要在 DSP_DDC_Configuration 之后进行调用。
示例	请参考 DSP_DDC_Execute()函数相关示例。

24.10 DSP_DDC_Execute

```
int DSP_DDC_Execute(
      void** DSP,
      const IQStream_TypeDef* IQStreamIn,
      IQStream_TypeDef* IQStreamOut
)
```

功能描述	
执行数字下变频（DDC）操作，将输入的 IQ 数据流通过当前 DDC 配置转换为下变频后的输出 IQ 数据流，用于后续处理或分析。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] IQStreamIn	输入 IQ 数据流的相关信息，包括 IQ 数据及相关配置信息。 详见 IQStream_TypeDef 结构体的定义。
[out] IQStreamOut	输出 IQ 数据流的相关信息，包括 IQ 数据及相关配置信息。 详见 IQStream_TypeDef 结构体的定义。

返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 DSP_DDC_Configuration 之后进行调用。

示例

```
int Status = -1; void* DSP = NULL; void* Device = NULL; int DeviceNum = 0;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
IQStream_TypeDef IQStreamOut;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = IQS_BusTriggerStop(&Device);
Status = DSP_Open(&DSP);
DSP_DDC_TypeDef DDC_ProfileIn, DDC_ProfileOut;
Status = DSP_DDC_DeInit(&DDC_ProfileIn);
uint32_t DDC_Points = 0;
Status = DSP_DDC_Configuration(&DSP, &DDC_ProfileIn, &DDC_ProfileOut);
uint32_t delay;
DSP_DDC_GetDelay(&DSP, &delay);
DSP_DDC_Reset(&DSP);
Status = DSP_DDC_Execute(&DSP, &IQStream, &IQStreamOut);
Status = DSP_Close(&DSP);
Status = Device_Close(&Device);
```

24.11 DSP_AudioAnalysis

void DSP_AudioAnalysis(const double Audio[], const uint64_t SamplePoints, const double SampleRate, DSP_AudioAnalysis_TypeDef* AudioAnalysis)
功能描述
对输入的音频数据进行分析，计算并输出音频电压（V）、音频频率（Hz）、信纳德（SINAD，dB）以及总谐波失真（THD，%）等关键音频性能参数。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] Audio[]	输入的音频数据数组，按时间顺序存储的音频采样值。
[in] SamplePoints	输入音频数据的采样点数，即 Audio 数组的长度。
[in] SampleRate	输入音频数据的采样率，单位为 Hz。
[out] AudioAnalysis	返回音频分析结果的结构体指针，包含音频电压、音频频率、SINAD 及 THD 等分析结果。 详见 DSP_AudioAnalysis_TypeDef 结构体的定义。
返回值	无。
调用约束	需要在 AM/FM 解调函数之后进行调用。

示例

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

```
ASD_Open(&AnalogMod);
bool reset = 1;
vector<float> result(StreamInfo.PacketSamples);
FM_DemodParam_TypeDef FM_DemodParam;
void* DSP = NULL;
DSP_Open(&DSP);
Status = IQS_BusTriggerStart(&Device);
DSP_AudioAnalysis_TypeDef  AudioAnalysis;
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = ADM_FMDemod(&AnalogMod, IQStream.AlternIQStream, Complex16bit, IQStream.IQS_StreamInfo.StreamSamples, IQStream.IQS_StreamInfo.IQSampleRate, reset, result.data());
vector<double> Audio(result.begin(), result.end());
DSP_AudioAnalysis(Audio.data(), StreamInfo.PacketSamples, StreamInfo.IQSampleRate, &AudioAnalysis);
Status = IQS_BusTriggerStop(&Device);
DSP_Close(&DSP);
ASD_Close(&AnalogMod);
Status = Device_Close(&Device);
```

24.12 DSP_LPF_DeInit

void DSP_LPF_DeInit(Filter_TypeDef* LPF_ProfileIn)	
功能描述	
初始化或重置低通滤波器（LPF）参数，为后续滤波处理准备，包括截止频率、阻带衰减等设置。	
兼容性	0.55.77 及之后版本支持
参数说明	
[out] LPF_ProfileIn	低通滤波器参数结构体，用于初始化或重置滤波器配置。 详见 Filter_TypeDef 结构体的定义。
返回值	无。
调用约束	在 DSP_LPF_Configuration 前调用。
示例	请参考 DSP_LPF_Execute_Real()函数相关示例。

24.13 DSP_LPF_Configuration

void DSP_LPF_Configuration(void** DSP, const Filter_TypeDef* LPF_ProfileIn, Filter_TypeDef* LPF_ProfileOut)	
功能描述	
配置低通滤波器（LPF）参数，根据输入的滤波器配置生成并应用对应的低通滤波器设置，同时返回实际生效的滤波器参数，用于后续 DSP 滤波处理。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] LPF_ProfileIn	输入的低通滤波器配置参数，用于生成滤波器系数。 详见 Filter_TypeDef 结构体的定义。
[out] LPF_ProfileOut	输出的低通滤波器实际配置参数，反映当前 DSP 中生效的 LPF 设置。 详见 Filter_TypeDef 结构体的定义。
返回值	无。
调用约束	需要在 DSP_LPF_DeInit 之后进行调用。
示例	请参考 DSP_LPF_Execute_Real()函数相关示例。

24.14 DSP_LPF_Reset

void DSP_LPF_Reset(void** DSP)	
功能描述	
重置低通滤波器（LPF）内部缓存与状态，用于清除历史滤波数据，避免对后续滤波结果产生影响，通常在重新开始数据处理或更换输入数据流前调用。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存空间引用，由 DSP_Open 返回，用于索引当前 DSP 实例并访问其 LPF 模块。
返回值	无。
调用约束	在 DSP_Open 后调用。
示例	请参考 DSP_LPF_Execute_Complex()函数相关示例。

24.15 DSP_LPF_Execute_Real

```
void DSP_LPF_Execute_Real(
        void** DSP,
        float Signal[],
        float LPF_Signal[]
)
```

功能描述

对实数信号执行低通滤波处理，使用当前已配置的低通滤波器参数对输入信号进行滤波，并输出滤波后的实数信号结果。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存引用，由 DSP_Open 返回，用于索引当前 DSP 实例。
[in] Signal[]	输入的实数信号数据数组，待进行低通滤波处理。
[out] LPF_Signal[]	输出的低通滤波后的实数信号数据数组，与输入信号长度一致。
返回值	无。
调用约束	在 DSP_LPF_Configuration 后调用。

示例

```
int Status = -1;
Sin_TypeDef NCO_Profile;
NCO_Profile.Amplitude = 10;
NCO_Profile.Frequency = 60e3;
NCO_Profile.Phase = 0;
NCO_Profile.SampleRate = 100e3;
NCO_Profile.Samples = 2000;
vector<float> sin(NCO_Profile.Samples);
vector<float> LPF_Signal(NCO_Profile.Samples);
void* DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn;
Filter_TypeDef LPF_ProfileOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
LPF_ProfileIn.As = 90;
LPF_ProfileIn.fc = 0.25;
```

```
LPF_ProfileIn.mu = 0;
LPF_ProfileIn.n = 90;
LPF_ProfileIn.Samples = NCO_Profile.Samples;
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileOut);
DSP_GenerateSineWaveform(sin.data(),&NCO_Profile);
DSP_LPF_Execute_Real(&DSP, sin.data(), LPF_Signal.data());
Status = DSP_Close(&DSP);
```

24.16 DSP_LPF_Execute_Complex

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

功能描述

对复数 IQ 信号执行低通滤波处理，使用当前已配置的低通滤波器参数对输入的 IQ 数据流进行滤波，并输出滤波后的复数 IQ 数据流结果。

兼容性	0.55.77 及之后版本支持
参数说明	
[in] DSP	DSP 内存空间引用，由 DSP_Open 返回，用于索引当前 DSP 实例及其低通滤波器配置。
[in] IQStreamIn	输入的复数 IQ 数据流，待进行低通滤波处理。 详见 IQStream_TypeDef 结构体的定义。
[out] IQStreamOut	输出的低通滤波后的复数 IQ 数据流。 详见 IQStream_TypeDef 结构体的定义。
返回值	无。
调用约束	在 DSP_LPF_Configuration 后调用。

示例

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

```
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream, IQStreamOut;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = IQS_BusTriggerStop(&Device);
void* DSP = NULL;
Status = DSP_Open(&DSP);
Filter_TypeDef LPF_ProfileIn, LPF_ProfileOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileOut);
DSP_LPF_Reset(&DSP);
DSP_LPF_Execute_Complex(&DSP, &IQStream, &IQStreamOut);
Status = DSP_Close(&DSP);
Status = Device_Close(&Device);
```

24.17 DSP_InterceptSpectrum

```
void DSP_InterceptSpectrum(
        const double StartFreq_Hz,
        const double StopFreq_Hz,
        const double Freq_Hz[],
        const float PowerSpec_dBm[],
        const uint32_t FullsweepTracePoints,
        double FrequencyOut[],
        float PowerSpecOut_dBm[],
        uint32_t* InterceptPoints
)
```

功能描述	
截取 SWP 模式下两边多余的频谱。	
兼容性	0.55.77 及之后版本支持
参数说明	
[in] StartFreq_Hz	指定需要截取的频谱片段的起始频率，单位：Hz。

[in] StopFreq_Hz	指定需要截取的频谱片段的终止频率，单位：Hz。
[in] Freq_Hz[]	被截取的频率数组，单位：Hz。
[in] PowerSpec_dBm[]	被截取的功率数组，单位：dBm。
[in] FullsweepTracePoints	被截取前的迹线点数。
[out] FrequencyOut[]	截取后的频率数组，单位：Hz。
[out] PowerSpecOut_dBm[]	截取后的功率数组，单位：dBm。
[out] InterceptPoints	实际截取到的有效迹线点数。
返回值	无。
调用约束	无。

示例

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
DeviceInfo_TypeDef DeviceInfo;
Status = Device_QueryDeviceInfo(&Device, &DeviceInfo);
SWP_Profile_TypeDef SWP_ProfileIn, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.StartFreq_Hz = 20000000;
SWP_ProfileIn.StopFreq_Hz = 300000000;
SWP_ProfileIn.RBW_Hz = 25000;
SWP_ProfileIn.RBWMode = RBW_Manual;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency_Full(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBmFull(TraceInfo.FullsweepTracePoints);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
int HopIndex = 0;
int FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
```

```
void* DSP = NULL;
DSP_Open(&DSP);
Status = SWP_GetFullSweep(&Device, Frequency_Full.data(), PowerSpec_dBmFull.data(),
&MeasAuxInfo);
uint32_t InterceptPoints;
DSP_InterceptSpectrum(SWP_ProfileOut.StartFreq_Hz, SWP_ProfileOut.StopFreq_Hz,
Frequency_Full.data(), PowerSpec_dBmFull.data(), TraceInfo.FullsweepTracePoints,
Frequency.data(), PowerSpec_dBm.data(), &InterceptPoints);
Frequency.resize(InterceptPoints);
PowerSpec_dBm.resize(InterceptPoints);
Status = Device_Close(&Device);
```

25. 结构体变量

25.1 BootProfile_TypeDef

BootProfile_TypeDef 详细定义	
PhysicalInterface_TypeDef PhysicalInterface	指定设备使用的物理总线类型，确定设备与主机的接口方式和通信类型。 详见 PhysicalInterface_TypeDef 枚举的定义。
DevicePowerSupply_TypeDef DevicePowerSupply	指定设备的供电方式，用于初始化设备电源接口。 详见 DevicePowerSupply_TypeDef 枚举的定义。
IPVersion_TypeDef ETH_IPVersion	指定 ETH 接口的 IP 协议版本，用于初始化网络通信。 详见 IPVersion_TypeDef 枚举的定义。
uint8_t ETH_IPAddress[16]	ETH 接口的 IP 地址，用于设备与主机的网络通信配置。
uint16_t ETH_RemotePort	ETH 接口的侦听端口号，用于网络连接和数据传输。
int32_t ETH_ErrorCode	ETH 接口的返回代码，用于指示网络初始化或通信状态。
int32_t ETH_ReadTimeOut	ETH 接口读取超时时间，单位毫秒(ms)，用于控制网络数据读取等待时间。

25.2 BootInfo_TypeDef

BootInfo_TypeDef 详细定义	
DeviceInfo_TypeDef DeviceInfo	设备基本信息结构体，包含设备唯一标识和各类版本信息。 详见 DeviceInfo_TypeDef 结构体的定义。
uint32_t BusSpeed	总线速度信息，用于表示设备通信总线的速率。
uint32_t BusVersion	总线固件版本号。
uint32_t APIVersion	API 版本号。
int ErrorCodes[7]	启动过程中的错误代码列表，每个元素表示一个错误类型。
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列表，每个元素表示一个警告类型。
int Warnings	启动过程中的警告总数。

25.3 DeviceInfo_TypeDef

DeviceInfo_TypeDef 详细定义	
uint64_t DeviceUID	设备唯一序列号，用于唯一标识设备。
uint16_t Model	设备类型或型号标识。
uint16_t HardwareVersion	设备硬件版本号。
uint16_t MFWVersion	设备 MCU 固件版本号。

uint16_t FFWVersion	设备 FPGA 固件版本号。
uint16_t PMUVersion	PMU 固件版本
uint16_t AGUVersion	AGU 固件版本

25.4 PowerSupplyState_TypeDef

PowerSupplyState_TypeDef 详细定义	
float rf_vlotage	射频板电源端口电压（v）
float rf_current	射频板电源端口电流（a）
float usb_vlotage	数字板 USB 端口电压（v）
float usb_current	数字板 USB 端口电流（a）

25.5 DeviceState_TypeDef

DeviceState_TypeDef 详细定义	
int16_t Temperature	设备温度，单位摄氏度，实际温度 = 0.01 × Temperature。
double AbsoluteTimeStamp	当前数据包对应的绝对时间戳。
float Latitude	当前数据包对应的纬度坐标，北纬为正，南纬为负。
float Longitude	当前数据包对应的经度坐标，东经为正，西经为负。
uint64_t nsSinceEpoch	当前数据包所对应的 ns 级系统时间戳。

25.6 NetworkDeviceInfo_TypeDef

NetworkDeviceInfo_TypeDef 详细定义	
uint64_t DeviceUID	设备序列号，用于唯一标识设备。
uint16_t Model	设备类型编号。
uint16_t HardwareVersion	硬件版本号。
uint32_t MFWVersion	MCU 固件版本号。
uint32_t FFWVersion	FPGA 固件版本号。
uint8_t IPAddress[4]	设备 IP 地址（IPv4）。
uint8_t SubnetMask[4]	设备子网掩码（IPv4）。

25.7 HardWareState_TypeDef

HardWareState_TypeDef 详细定义	
GNSSPeriphType_TypeDef GNSSPeriphType	GNSS 外设类型。 详见 GNSSPeriphType_TypeDef 枚举的定义。
GNSSType_TypeDef GNSSType	GNSS 接收机类型。 详见 GNSSType_TypeDef 枚举的定义。
OCXOType_TypeDef OCXOType	GNSS 上的 OCXO 类型。 详见 OCXOType_TypeDef 枚举的定义。
uint8_t InternalOCXO	设备内部参考时钟是否为恒温晶振。
uint8_t SignalSourceEn	设备是否支持信号源功能。
uint8_t ADC_VariableRateEn	设备是否支持 ADC 可变采样率。
uint8_t IM3_filter	补充中频滤波器（IM3 增强）。 0：未启用，1：启用。

25.8 GNSSInfo_TypeDef

GNSSInfo_TypeDef 详细定义	
float latitude	返回当前定位的纬度坐标。
float longitude	返回当前定位的经度坐标。
int16_t altitude	返回当前定位的海拔高度。
uint8_t SatsNum	返回当前参与定位的卫星数。
uint8_t GNSS_LockState	返回 GNSS 锁定状态。 0：未锁定，1：锁定。
uint8_t DOCXO_LockState	返回 DOCXO 锁定状态。 0：未锁定，1：锁定。
DOCXOWorkMode_TypeDef DOCXO_WorkMode	返回 DOCXO 工作状态。 详见 DOCXOWorkMode_TypeDef 枚举的定义。
GNSSAntennaState_TypeDef GNSSAntennaState	返回天线状态。 详见 GNSSAntennaState_TypeDef 枚举的定义。
int16_t hour	返回 GNSS 时间和日期信息中的小时部分。
int16_t minute	返回 GNSS 时间和日期信息中的分钟部分。
int16_t second	返回 GNSS 时间和日期信息中的秒钟部分。
int16_t Year	返回 GNSS 时间和日期信息中的年份部分。
int16_t month	返回 GNSS 时间和日期信息中的月份部分。
int16_t day	返回 GNSS 时间和日期信息中的日期部分。

25.9 GNSS_SatDate_TypeDef

GNSS_SatDate_TypeDef 详细定义	
uint8_t SatsNum_All	返回当前范围内可视卫星的数量。
uint8_t SatsNum_Use	返回用于定位的卫星数量。
GNSS_SNR_TypeDef GNSS_SNR_UsePos	返回用于定位的卫星信噪比信息，包括最大、最小和平均信噪比。 详见 GNSS_SNR_TypeDef 结构体的定义。
GNSS_SNR_TypeDef GNSS_SNR_NotUsePos	返回在视野内但未用于定位的卫星信噪比信息，如最大、最小和平均信噪比。 详见 GNSS_SNR_TypeDef 结构体的定义。

25.10 GNSS_SNR_TypeDef

GNSS_SNR_TypeDef 详细定义	
uint8_t Max_SatxC_No	当前信号集合中的最大信噪比。
uint8_t Min_SatxC_No	当前信号集合中的最小信噪比。
uint8_t Avg_SatxC_No	当前信号集合的平均信噪比。

25.11 SWP_Profile_TypeDef

SWP_Profile_TypeDef 详细定义	
double StartFreq_Hz	起始频率，单位 Hz，范围：9kHz ~ (设备频率上限-100Hz)
double StopFreq_Hz	终止频率，单位 Hz，范围：（9kHz + 100 Hz）~ 设备频率上限
double CenterFreq_Hz	中心频率，单位 Hz，范围：9kHz ~ StopFreq_Hz-50Hz
double Span_Hz	频率扫宽，单位 Hz，范围：≥ 100 Hz
double RefLevel_dBm	参考电平，单位 dBm，范围：-50dBm ~ 23dBm
double RBW_Hz	分辨率带宽，单位 Hz。范围：0.1 ~ 10 MHz
double VBW_Hz	视频带宽，单位 Hz。范围：0.1 ~ 10 MHz
double SweepTime	当扫描时间模式指定为Manual时，该参数为绝对时间； 当指定为*N时，该参数为扫描时间倍率。范围：0.1s ~ 9999s
SWP_FreqAssignment_TypeDef FreqAssignment	设置频率的指定方式，选择以 StartStop 或 CenterSpan 设定频率。 默认使用 StartStop 方式。 详见 SWP_FreqAssignment_TypeDef 枚举的定义。
Window_TypeDef Window	指定FFT分析所使用的窗函数，默认使用Blackman_Nuttall窗。 详见Window_TypeDef枚举的定义。
RBWMode_TypeDef RBWMode	设置RBW更新方式，默认使用RBW_Auto。 详见RBWMode_TypeDef枚举的定义。

VBWMode_TypeDef VBWMode	设置VBW更新方式，默认使用VBW_TenTimesRBW。 详见VBWMode_TypeDef枚举的定义。
SweepTimeMode_TypeDef SweepTimeMode	设置扫描时间模式，默认使用SWTMode_minSWT模式。 详见SweepTimeMode_TypeDef枚举的定义。
Detector_TypeDef Detector	设置检波器，默认使用Detector_PosPeak检波器。 详见Detector_TypeDef枚举的定义。
TraceFormat_TypeDef TraceFormat	设置迹线格式，默认使用TraceFormat_Standard迹线格式。 详见TraceFormat_TypeDef枚举的定义。
TraceDetectMode_TypeDef TraceDetectMode	设置迹线检波模式（频率轴），默认使用TraceDetectMode_Auto。 详见TraceDetectMode_TypeDef枚举的定义。
TraceDetector_TypeDef TraceDetector	设置迹线检波器类型，默认使用TraceDetector_AutoSample 若需指定检波器，先将TraceDetectMode设置为TraceDetectMode_Manual。 详见TraceDetector_TypeDef枚举的定义。
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，默认使用SweepSpeedPreferred策略。 详见TracePointsStrategy_TypeDef枚举的定义。
TraceAlign_TypeDef TraceAlign	设置迹线对齐的方式，默认使用NativeAlign方式。 详见TraceAlign_TypeDef枚举的定义。
FFTExecutionStrategy_TypeDef FFTExecutionStrategy	设置FFT执行策略，默认使用Auto策略。 详见FFTExecutionStrategy_TypeDef枚举的定义。
RxPort_TypeDef RxPort	设置信号接收端口，仅终止频率为8.5 GHz及以下的仪器支持。 详见RxPort_TypeDef枚举的定义。
SpurRejection_TypeDef SpurRejection	设置杂散抑制，默认采用Standard，杂散抑制等级越高，扫描速率越慢 详见SpurRejection_TypeDef枚举的定义。
ReferenceClockSource_TypeDef ReferenceClockSource	设置参考时钟源，10 MHz。 详见ReferenceClockSource_TypeDef枚举的定义。
double ReferenceClockFrequency	设置参考时钟频率 Hz（仅可设置10 MHz），可手动对系统时钟准确度进行修正。

uint8_t EnableReferenceClockOut	设置使能参考时钟输出，仅终止频率为9 GHz及以上的设备支持10 MHz参考时钟输出。 0：关闭；1：开启，输出10 MHz参考时钟
SystemClockSource_TypeDef SystemClockSource	设置系统时钟源，默认使用内部系统时钟。 详见SystemClockSource_TypeDef枚举的定义。
double ExternalSystemClockFrequency	外部系统时钟频率，Hz 当接入外部系统时钟源时，需将此参数设置为10 MHz
SWP_TriggerSource_TypeDef TriggerSource	设置接收机的扫频输入触发源，默认使用内部触发自由运行。 详见SWP_TriggerSource_TypeDef枚举的定义。
TriggerEdge_TypeDef TriggerEdge	设置输入触发边沿，默认由上升沿触发。 详见TriggerEdge_TypeDef枚举的定义。
TriggerOutMode_TypeDef TriggerOutMode	设置触发输出的模式，默认无触发输出。 详见TriggerOutMode_TypeDef枚举的定义。
TriggerOutPulsePolarity_TypeDef TriggerOutPulsePolarity	设置触发输出的脉冲极性，默认设置正脉冲。 详见TriggerOutPulsePolarity_TypeDef枚举的定义。
uint32_t PowerBalance	设置SWP模式下的动态功耗控制，默认为0。 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：默认使用低噪声。 详见GainStrategy_TypeDef枚举的定义。
PreamplifierState_TypeDef Preamplifier	设置前置放大器动作，默认为自动使能。 详见PreamplifierState_TypeDef枚举的定义。
uint8_t AnalogIFBWGrade	设置中频带宽档位，默认使用声表滤波器。 0：声表滤波器，带宽100 MHz，20%滤波边带，具有较好的带内平坦度； 1：LC滤波器，带宽110 MHz，10%滤波边带，具有较好的带外抑制性能。
uint8_t IFGainGrade	设置中频增益档位，默认为2档。 终止频率8.5 GHz及以下设备，范围：0 ~ 11 终止频率9.0 GHz及以上设备，范围：0 ~ 3
int8_t Atten	设置衰减dB，设定频谱仪通道衰减量，默认-1（自动）。 范围：-1 ~ 33，当该值不等于-1（自动）时，其优先于RefLevel_dBm，此时参考电平 = 衰减 - 10
uint8_t EnableIFAGC	特定设备适用。中频AGC控制，0：AGC关闭，使用MGC方式；1：AGC开启,当中频饱和时降低中频增益避免饱和。

SWP_TraceType_TypeDef TraceType	设置输出迹线类型，默认输出正常迹线 详见SWP_TraceType_TypeDef枚举的定义。
LOOptimization_TypeDef LOOptimization	设置本振优化，默认使用自动方式。 详见LOOptimization_TypeDef枚举的定义。

25.12 SWP_TraceInfo_TypeDef

SWP_TraceInfo_TypeDef 详细定义	
int FullsweepTracePoints	返回当前配置下，完整迹线的点数。
int PartialsweepTracePoints	返回当前配置下，每个频点的迹线点数，即每次GetPart的点数。
int TotalHops	返回当前配置下，完整迹线的频点数，即一条完整迹线需要GetPart的次数。
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最近的频率点。
uint32_t UserStopIndex	迹线数组中与用户指定StopFreq_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	返回当前配置下，迹线两点间的频率间隔。
double StartFreq_Hz	迹线第一个频点的频率。
double AnalysisBW_Hz	每个频点对应的分析带宽。
int TraceDetectRatio	迹线检波比，范围：[1，2^32-1] 定义了原始采样点与输出迹线点之间的映射比例。迹线检波器从原始频谱序列中，每隔TraceDetectRatio个数据点提取一个或两个有效数据点，作为输出迹线的组成部分。
int DecimateFactor	时域数据的抽取倍数。
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。
double FrameTime	帧扫描时间：用于进行单帧FFT分析的信号持续时间（单位：S）
double EstimateMinSweepTime	当前配置下，所能设定的最小扫描时间 单位：S，结果主要受Span、RBW、VBW、帧扫描时间等因素影响。

DataFormat_TypeDef DataFormat	时域数据格式。 详见DataFormat_TypeDef枚举的定义。
uint64_t SamplePoints	时域数据采样长度。
uint32_t GainParameter	增益相关参数，其中第3个字节表示前置放大器状态 PreAmplifierState(23~16Bit) 0：关闭；1：开启
DSPPlatform_Typedef DSPPlatform	返回当前配置所使用的DSP运算平台。 详见DSPPlatform_TypeDef枚举的定义。

25.13 MeasAuxInfo_TypeDef

MeasAuxInfo_TypeDef 详细定义	
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。
double SysTimeStamp	系统时间戳，单位 s。 设备上电后，设备内部计数器以8ns间隔为周期开始计数。
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。
float Altitude	当前数据包对应的海拔，单位米。
float SATHealth	当前GNSS定位卫星的健康度（SNR）。
double IFAGCGain	当前的IFAGC的增益，正值代表放大，负值代表衰减，单位dB。
double RefClkFreqOffset	表示当前设备参考时钟频率的偏移，该值以GNSS频率为参考，单位ppm。
uint64_t nsSinceEpoch	当前数据包所对应的ns级系统时间戳。

25.14 SWPTrace_TypeDef

SWPTrace_TypeDef 详细定义	
double* Freq_Hz	指向频率数组的首地址。
float* PowerSpec_dBm	指向功率数组的首地址。

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。无单位的原始IQ数据
float ScaletoV	时域数据至电压绝对值（V）的系数。 原始IQ数据* ScaletoV 为以V为单位的IQ数据
MeasAuxInfo_TypeDef MeasAuxInfo	详见MeasAuxInfo_TypeDef结构体的定义。
SWP_Profile_TypeDef SWP_Profile	详见SWP_Profile_TypeDef结构体的定义。
SWP_TraceInfo_TypeDef SWP_TraceInfo	详见SWP_TraceInfo_TypeDef结构体的定义。
DeviceInfo_TypeDef DeviceInfo	详见DeviceInfo_TypeDef结构体的定义。
DeviceState_TypeDef DeviceState	详见DeviceState_TypeDef结构体的定义。

25.15 PNM_Profile_TypeDef

PNM_Profile_TypeDef 详细定义	
double CenterFreq	设置基波的中心频率，单位Hz。
float Threshold	设置载波判决门限，单位dBm，高于该门限载波被识别。
double RBWRatio	RBW比例（各段RBW/各段起始频率），范围：0.01 ~ 0.3
double StartOffsetFreq	起始频偏，范围：1 Hz ~ 9 MHz
double StopOffsetFreq	终止频偏，范围：10 Hz ~ 10 MHz
uint32_t TraceAverage	迹线平滑次数。

25.16 PNM_MeasInfo_TypeDef

PNM_MeasInfo_TypeDef 详细定义	
uint32_t Segments	频率分段数（十倍频程划分）。
uint32_t TracePoints	相位噪声测量结果的总迹线点数。
uint32_t PartialUpdateCounts	单次相位噪声分析过程中，对应的迹线刷新次数，即Get类接口的调用次数。
uint32_t FramesInSegment [PHASENOISE_MAXFREQSEGMENT]	各分段的总帧数，PHASENOISE_MAXFREQSEGMENT定义了频率分段（十倍频程段）的最大数量，总数上限为7。

uint32_t FrameDetRatioOfSegment [PHASENOISE_MAXFREQSEGMENT]	各分段对应的帧检波比。

double StartFreqOfSegment [PHASENOISE_MAXFREQSEGMENT]	各分段对应的起始频率。
double StopFreqOfSegment [PHASENOISE_MAXFREQSEGMENT]	各分段对应的终止频率。
double RBWOfSegment [PHASENOISE_MAXFREQSEGMENT]	各分段对应的RBW，单位Hz。

25.17 IQS_Profile_TypeDef

IQS_Profile_TypeDef 详细定义	
double CenterFreq_Hz	中心频率，范围：9kHz ~ (设备最高频率 - 50 MHz）。
double RefLevel_dBm	参考电平，范围：-50 dBm ~ 23 dBm。
uint32_t DecimateFactor	时域数据的抽取倍数，范围：1 ~ 4096。
RxPort_TypeDef RxPort	设置信号接收端口，仅终止频率 8.5GHz 及以下，选配内置信号源选件的设备适用。 详见 RxPort_TypeDef 枚举的定义。
uint32_t BusTimeout_ms	设置数据传输超时时间（ms），范围：1ms ~ 10 s。 若获取 IQ 流函数在指定时间内未获取到数据，则视为超时并返回错误。
IQS_TriggerSource_TypeDef TriggerSource	设置输入触发源，默认使用总线触发。 详见 IQS_TriggerSource_TypeDef 枚举的定义。
TriggerEdge_TypeDef TriggerEdge	设置输入触发边沿，默认使用上升沿。 详见 TriggerEdge_TypeDef 枚举的定义。
TriggerMode_TypeDef TriggerMode	设置输入触发模式，默认使用 FixedPoints。 详见 TriggerMode_TypeDef 枚举的定义。
uint64_t TriggerLength	设置触发长度，即每次触发所采集的数据点数。仅在 TriggerMode = FixedPoints 时生效 范围：32 ~ (UINT64_MAX/6)。 若使用 Bus 触发，在获取 IQ 数据格式为 16bit 时，若 TriggerLength 不超过 33263616 个点，即可在两次触发之间先获取数据再处理数据。
TriggerOutMode_TypeDef TriggerOutMode	设置触发输出模式，默认使用上升沿。 详见 TriggerOutMode_TypeDef 枚举的定义。

TriggerOutPulsePolarity_TypeDef TriggerOutPulsePolarity	设置触发输出脉冲极性，默认使用上升沿。 详见 TriggerOutPulsePolarity_TypeDef 枚举的定义。
double TriggerLevel_dBm	设置电平触发门限。 仅在 TriggerSource = Level 时生效，范围：-70 dBm ~ 参考电平。
double TriggerLevel_SafeTime	电平触发防抖安全时间，单位 s，仅在 TriggerSource = Level 时生效， 范围：0 s ~ $(2^{32}-1)*(1.0/125MHz)$ s。 若触发信号的有效电平持续时间短于设定值，则触发无效。
double TriggerDelay	设置触发延迟，单位 s。 触发后，触发动作将在此延时之后执行。 范围：0 s ~ $(2^{32}-1)*(1.0/125MHz)$ s。
double PreTriggerTime	设置预触发时间，单位 s。 范围：0 s ~ $(2^{16}-1)*(1.0/125MHz)$ s。 触发发生时，将同步保存触发时刻前该时长内的数据。
TriggerTimerSync_TypeDef TriggerTimerSync	设置触发定时器的同步，默认使用与外触发上升沿同步。 详见 TriggerTimerSync_TypeDef 枚举的定义。
double TriggerTimer_Period	设置定时触发周期，单位秒。仅在 TriggerSource = Timer 时生效， 范围：0 s ~ $(2^{32}-1)*(1.0/125MHz)$ s。
uint8_t EnableReTrigger	自动重触发，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，0：禁用，1：启用。
double ReTrigger_Period	自动重触发，设置每次主触发后，再触发的触发周期，单位 s。 范围：0 s ~ $(2^{32}-1)*(1.0/125MHz)$ s。
uint16_t ReTrigger_Count	自动重触发，设置每次主触发后，再触发的执行次数。 范围：0 次 ~ $(2^{16}-1)$ 次。
DataFormat_TypeDef DataFormat	设置 IQ 数据的输出数据格式，默认使用 16 位格式。 详见 DataFormat_TypeDef 枚举的定义。
GainStrategy_TypeDef GainStrategy	设置增益策略，默认使用低噪声策略。 详见 GainStrategy_TypeDef 枚举的定义。
PreamplifierState_TypeDef Preamplifier	设置前置放大器动作，默认自动使能前置放大器。 详见 PreamplifierState_TypeDef 枚举的定义。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。 0：声表滤波器，带宽 100 MHz，20%滤波边带，较好的带内平坦度； 1：LC 滤波器，带宽 110 MHz，10%滤波边带，较好的带外抑制性能。

uint8_t IFGainGrade	设置中频增益档位。档位越高中频增益越高。 对于终止频率 8.5 GHz 及以下仪器，挡位范围为：0 ~ 11。 对于终止频率 20 GHz 或 40 GHz 的仪器，挡位范围为：0 ~ 3。
uint8_t EnableDebugMode	调试模式，高级应用不推荐用户自行使用，默认值为 0。
ReferenceClockSource_TypeDef ReferenceClockSource	设置参考时钟源，默认使用 10 MHz 内部时钟源。 详见 ReferenceClockSource_TypeDef 枚举的定义。
double ReferenceClockFrequency	设置参考时钟频率，单位：Hz。 仅支持 10 MHz 参考时钟频率。
uint8_t EnableReferenceClockOut	设置使能参考时钟输出，仅终止频率 9G 及以上设备支持。 0：不输出；1：输出 10 MHz 参考时钟。
SystemClockSource_TypeDef SystemClockSource	设置系统时钟源。 详见 SystemClockSource_TypeDef 枚举的定义。
double ExternalSystemClockFrequency	外部系统时钟频率，单位：Hz。
double NativeIQSampleRate_SPS	设置原生的 IQ 采样速率，设备可通过调整该参数对采样率进行调整； 若调整此参数后未生效，则表示此设备不支持调整设备采样率。
uint8_t EnableIFAGC	中频 AGC 控制，0：AGC 关闭，使用 MGC 方式；1：AGC 开启，当中频饱和时降低中频增益避免饱和。 若调整此参数后未生效，则表示此设备不支持此功能。
int8_t Atten	设置衰减 dB，设定频谱仪通道衰减量，默认-1（自动）。 范围：-1 ~ 33。 当该值不等于-1（自动）时，其优先于参考电平（RefLevel_dBm），此时，参考电平将自动设置为 衰减值 − 10 dBm。
DCCancelerMode_TypeDef DCCancelerMode	特定设备适用。设置直流抑制器模式，默认开启高通滤波器。 若设备在中心频率处出现直流分量，请开启此功能抑制直流分量，若未出现直流分量，则无需设置此参数。 详见 DCCancelerMode_TypeDef 枚举的定义。
QDCMode_TypeDef QDCMode	特定设备适用。设置 IQ 幅相修正器模式。 若开启 QDC 功能后，IQ 数据的幅相特征未发生变化，则此设备不需开启 QDC 功能。 详见 QDCMode_TypeDef 枚举的定义。
float QDCIGain	特定设备适用。设置 I 通道的归一化线性增益。 1.0 表示无增益，设置范围：0.8 ~ 1.2。

	仅在 QDCMode = QDCManualMode 时生效。
float QDCQGain	特定设备适用。设置 Q 通道的归一化线性增益。 1.0 表示无增益，设置范围：0.8 ~ 1.2。 QDCMode = QDCManualMode 时生效。
float QDCPhaseComp	特定设备适用。设置相位补偿系数。 设置范围：-0.2~+0.2。 仅在 QDCMode = QDCManualMode 时生效。
int8_t DCCIOffset	特定设备适用。设置 I 通道直流偏置，LSB。 仅在 DCCancelerMode = DCCManualOffsetMode 时生效。
int8_t DCCQOffset	特定设备适用。设置 Q 通道直流偏置，LSB。 仅在 DCCancelerMode = DCCManualOffsetMode 时生效。
LOOptimization_TypeDef **LOOptimization**	设置本振优化，默认使用自动模式。 详见 LOOptimization_TypeDef 枚举的定义。

25.18 IQS_StreamInfo_TypeDef

IQS_StreamInfo_TypeDef 详细定义	
double Bandwidth	当前配置下，接收机物理通道或数字信号处理的带宽。
double IQSampleRate	当前配置下，对应的 IQ 单路采样率，单位 S/s（Sample/second）。
uint64_t PacketCount	当前配置单帧（Frame）对应的总数据包数，仅在 TriggerMode = Fixedpoints 模式下生效。
uint64_t StreamSamples	TriggerMode = Fixedpoints 时：表示当前配置单帧（Frame）对应采样的总点数； TriggerMode = Adaptive 时：该值无物理意义，返回值为 0。
uint64_t StreamDataSize	TriggerMode = Fixedpoints 时：表示当前配置单帧（Frame）对应采样的总字节数； TriggerMode = Adaptive 时：该值无物理意义，返回值为 0。
uint32_t PacketSamples	每次调用 IQS_GetIQStream 时，获取到的数据包中包含的采样点数。
uint32_t PacketDataSize	每次调用 IQS_GetIQStream 时，获取的有效数据字节数。
uint32_t GainParameter	增益相关参数，其中第 3 个字节表示前置放大器状态。 PreAmplifierState(23~16Bit)，0：关闭，1：开启。

25.19 IQS/DET/RTA_TriggerInfo_TypeDef

IQS_TriggerInfo_TypeDef / DET_TriggerInfo_TypeDef / RTA_TriggerInfo_TypeDef 详细定义	
uint64_t SysTimerCountOfFirstDataPoint	数据包中首个数据点对应的系统时间计数器值。 设备上电启动后，内部计时器以 8ns 为周期开始计数，返回的系统时间戳即为此时计数器的值
uint16_t InPacketTriggeredDataSize	数据包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]	数据包中触发沿相对包首点的偏移。
uint64_t SysTimerCountOfEdges[25]	数据包中各触发边沿的对应的系统时间戳。
int8_t EdgeType[25]	数据包中各触发边沿的极性。

25.20 IQStream_TypeDef

IQStream_TypeDef 详细定义	
void* AlternIQStream	返回时域 IQ 数据包（包内数据按 IQIQIQ...的排列方式进行存储）。 一整个数据包为固定 64968 个字节。 IQ 数据类型设置为 int8_t 时，I、Q 两路分别为 32484 个点，每个点占 1 个字节； IQ 数据类型设置为 int16_t 时，I、Q 两路分别为 16242 个点，每个点占 2 个字节； IQ 数据类型设置为 int32_t 时，I、Q 两路分别为 8121 个点，每个点占 4 个字节。
float IQS_ScaleToV	将 ADC 采集的原始时域数据转换为电压绝对值（V）的系数。
float MaxPower_dBm	输出当前数据包中最大功率值，单位：dBm。
uint32_t MaxIndex	功率最大值在数据包中的索引，即该值对应的采样点的位置。
IQS_Profile_TypeDef IQS_Profile	数据的配置信息。 详见 IQS_Profile_TypeDef 结构体的定义。
IQS_StreamInfo_TypeDef StreamInfo	数据的格式信息。 详见 IQS_StreamInfo_TypeDef 结构体的定义。
IQS_TriggerInfo_TypeDef TriggerInfo	数据的触发信息。 详见 IQS_TriggerInfo_TypeDef 结构体的定义。
DeviceInfo_TypeDef DeviceInfo	数据的设备信息。 详见 DeviceInfo_TypeDef 结构体的定义。
DeviceState_TypeDef DeviceState	数据的设备状态信息。 详见 DeviceState_TypeDef 结构体的定义。

25.21 DET_Profile_TypeDef

DET_Profile_TypeDef 详细定义	
double CenterFreq_Hz	请参考 IQS_Profile_TypeDef 结构体同名参数。
double RefLevel_dBm	
uint32_t DecimateFactor	
RxPort_TypeDef RxPort	
uint32_t BusTimeout_ms	
DET_TriggerSource_TypeDef TriggerSource	
TriggerEdge_TypeDef TriggerEdge	
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	
Detector_TypeDef Detector	设置检波器，默认使用取样检波。 详见 Detector_TypeDef 枚举的定义。
uint16_t DetectRatio	设置 DET 迹线检波比，检波器对功率迹线进行检波，每 DetectRatio 个原始数据点检出为 1 个输出迹线点。
GainStrategy_TypeDef GainStrategy	请参考 IQS_Profile_TypeDef 结构体同名参数。

PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
uint8_t EnableDebugMode	
ReferenceClockSource_TypeDef ReferenceClockSource	
double ReferenceClockFrequency	
uint8_t EnableReferenceClockOut	
SystemClockSource_TypeDef SystemClockSource	
double ExternalSystemClockFrequency	
int8_t Atten	
uint8_t EnableIFAGC	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
LOOptimization_TypeDef LOOptimization	

25.22 DET_StreamInfo_TypeDef

DET_StreamInfo_TypeDef 详细定义	
uint64_t PacketCount	当前配置下单次触发采集的数据包总数，仅在 TriggerMode = Fixedpoints 模式下生效。
uint64_t StreamSamples	TriggerMode = Fixedpoints 时：表示当前配置单帧（Frame）对应采样的总点数； TriggerMode = Adaptive 时：该值无物理意义，返回值为 0。

uint64_t StreamDataSize	TriggerMode = Fixedpoints 时：表示当前配置单帧（Frame）对应采样的总字节数； TriggerMode = Adaptive 时：该值无物理意义，返回值为 0。
uint32_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。
uint32_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	增益相关参数，其中第 3 个字节表示前置放大器状态。 PreAmplifierState(23~16Bit)，0：关闭，1：开启。

25.23 ZSP_Profile_TypeDef

ZSP_Profile_TypeDef 详细定义	
double CenterFreq_Hz	请参考 IQS_Profile_TypeDef 结构体同名参数。
double RefLevel_dBm	
uint32_t DecimateFactor	
RxPort_TypeDef RxPort	
uint32_t BusTimeout_ms	
DET_TriggerSource_TypeDef TriggerSource	
TriggerEdge_TypeDef TriggerEdge	
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	
Detector_TypeDef Detector	
uint16_t DetectRatio	
GainStrategy_TypeDef GainStrategy	
PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
uint8_t EnableDebugMode	
ReferenceClockSource_TypeDef ReferenceClockSource	
double ReferenceClockFrequency	
uint8_t EnableReferenceClockOut	
SystemClockSource_TypeDef SystemClockSource	
double ExternalSystemClockFrequency	
int8_t Atten	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	
float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
LOOptimization_TypeDef LOOptimization	
double RBW_Hz	分辨率带宽，单位：Hz。
double VBW_Hz	视频带宽，单位：Hz。

VBWMode_TypeDef **VBWMode**	VBW 更新方式。 详见 VBWMode_TypeDef 枚举的定义。
RBWFilterType_TypeDef **RBWFilterType**	RBW 滤波器类型。 详见 RBWFilterType_TypeDef 枚举的定义。

25.24 RTA_Profile_TypeDef

RTA_Profile_TypeDef 详细定义	
double **CenterFreq_Hz**	请参考 SWP_Profile_TypeDef 结构体同名参数。
double **RefLevel_dBm**	
double **RBW_Hz**	
double **VBW_Hz**	
RBWMode_TypeDef **RBWMode**	
VBWMode_TypeDef **VBWMode**	
uint32_t **DecimateFactor**	设置抽取倍数。范围：2^n (n = 1 ~ 12)。
Window_TypeDef **Window**	请参考 SWP_Profile_TypeDef 结构体同名参数。
SweepTimeMode_TypeDef **SweepTimeMode**	
double **SweepTime**	
Detector_TypeDef **Detector**	
TraceDetectMode_TypeDef **TraceDetectMode**	
TraceDetector_TypeDef **TraceDetector**	
uint32_t **TraceDetectRatio**	迹线检波检波比，范围：[1，2^{32}-1]。 定义了原始采样点与输出迹线点之间的映射比例。迹线检波器从原始频谱序列中，每隔 TraceDetectRatio 个数据点提取一个有效数据点，作为输出迹线的组成部分。
RxPort_TypeDef **RxPort**	请参考 IQS_Profile_TypeDef 结构体同名参数。
uint32_t **BusTimeout_ms**	
RTA_TriggerSource_TypeDef **TriggerSource**	
TriggerEdge_TypeDef **TriggerEdge**	
TriggerMode_TypeDef **TriggerMode**	

double TriggerAcqTime	定义单次触发事件后数据采集的时长，单位 s。范围：≥ 256 ns。 仅在 TriggerMode = Fixedpoints 模式下生效。
TriggerOutMode_TypeDef TriggerOutMode	请参考 IQS_Profile_TypeDef 结构体同名参数。
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 GainStrategy	
PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
uint8_t EnableDebugMode	
ReferenceClockSource_TypeDef ReferenceClockSource	
double ReferenceClockFrequency	
uint8_t EnableReferenceClockOut	
SystemClockSource_TypeDef SystemClockSource	
double ExternalSystemClockFrequency	
int8_t Atten	
uint8_t EnableIFAGC	
DCCancelerMode_TypeDef DCCancelerMode	
QDCMode_TypeDef QDCMode	
float QDCIGain	

float QDCQGain	
float QDCPhaseComp	
int8_t DCCIOffset	
int8_t DCCQOffset	
LOOptimization_TypeDef LOOptimization	

25.25 RTA_FrameInfo_TypeDef

RTA_FrameInfo_TypeDef 详细定义	
double StartFrequency_Hz	频谱的起始频率，单位：Hz。
double StopFrequency_Hz	频谱的终止频率，单位：Hz。
double POI	100%截获概率下的信号最短持续时间，单位：s。
double TraceTimestampStep	每包数据内各条迹线的时间戳步进。(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，单位：s。
uint32_t PacketCount	当前配置对应的总数据包数，仅在 TriggerMode = Fixedpoints 模式下生效。
uint32_t PacketFrame	每个数据包中的有效帧数。
uint32_t FFTSize	每帧进行 FFT 时采用的点数。
uint32_t FrameWidth	FFT 帧截取后的有效点数，对应数据包中每条迹线的显示点数，可作为概率密度图的 X 轴点数(宽度)。
uint32_t FrameHeight	FFT 帧对应的频谱幅度范围，可作为概率密度图的 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：PacketFrame × FrameWidth。
uint32_t MaxDensityValue	概率密度位图的单个像素点的最大累计值。
uint32_t GainParameter	增益相关参数，其中第 3 个字节表示前置放大器状态。 PreAmplifierState(23~16Bit)，0：关闭；1：开启

25.26 RTA_PlotInfo_TypeDef

RTA_PlotInfo_TypeDef 详细定义	
float ScaleTodBm	由线性功率转对数功率导致的压缩。Trace 的绝对功率等于 SpectrumStream[] * ScaleTodBm + OffsetTodBm。
float OffsetTodBm	相对功率转换为绝对功率的偏移。bitmap 的绝对功率轴范围(Y 轴)等于 FrameHeight * ScaleTodBm + OffsetTodBm。

uint64_t SpectrumBitmapIndex	概率密度图的获取次数，当用户需要多张概率密度图叠加时，可使用该元素作为叠加索引。

25.27 Demod_Profile_TypeDef

Demod_Profile_TypeDef 详细定义	
uint64_t SamplePoints	采样点数。 范围：[16384，320000]，建议设置为 2000*采样率/符号率，若解调 128QAM 和 256QAM 信号，建议设置为 4000*采样率/符号率。
double SampleRate	采样率，Hz。
double SymbolRate	符号率，sym/s。范围：[采样率/64，采样率/4]。
Demod_ModType_TypeDef ModType	设置待解调信号的调制类型，支持 2ASK/2FSK/4FSK/GMSK/BPSK/QPSK/8PSK/16QAM/32QAM/64QAM/128QAM/256QAM。
Demod_FilterType_TypeDef FilterType	设置滤波器类型，目前仅支持根升余弦滤波器。 详见 Demod_FilterType_TypeDef 枚举的定义。
double FilterAlpha	滤波器滚降系数，范围：[0.01，0.99]。

25.28 DemodInfo_TypeDef

DemodInfo_TypeDef 详细定义	
double* eDiagram	眼图数据起始内存地址。
uint32_t eDiagram_Len	眼图数据长度。
double* I_constellation	I 路星座图数据起始内存地址。
double* Q_constellation	Q 路星座图数据起始内存地址。
uint32_t constellation_Len	星座图数据长度。
int32_t* bitStream	比特流数据起始内存地址。
uint32_t bitStream_Len	比特流数据长度。
int32_t* symbol	码流数据起始内存地址。
uint32_t symbol_Len	码流数据长度。
double* EVM	EVM 数据起始内存地址，同为 FSK Error、ASK Error
uint32_t EVM_Len	EVM 数据长度。
double EVM_RMS	均方根 EVM，%。
double EVM_MAX	峰值 EVM，%。
double* PhaseError	相位误差数据起始内存地址，单位角度。
uint32_t PhaseError_Len	相位误差数据长度。
double PhaseError_RMS	均方根相位误差，单位角度。

double PhaseError_MAX	峰值相位误差，单位角度。
double* MagError	幅度误差数据起始内存地址。
uint32_t MagError_Len	幅度误差数据长度。
double MagError_RMS	均方根幅度误差，%。
double MagError_MAX	峰值幅度误差，%。
double FreqError	频率误差，载波相对于中心频率的频率误差，同为 CarrFreqOffset，单位 Hz
double IQ_Offset	IQ 偏移，单位 dB，仅 PSK 和 QAM 模式下有效。
double SNR	信噪比，单位 dB，仅 PSK 和 QAM 模式下有效。
double GainImb	IQ 增益不平衡，单位 dB，仅 PSK 和 QAM 模式下有效。
double QuadError	IQ 正交倾斜误差，单位角度，仅 PSK 和 QAM 模式下有效。
double FSK_Deviation	FSK 频偏，单位 Hz。
double CarrPower	载波功率，单位 dBm，仅 ASK 模式下有效。
double ASK_Depth	ASK 调制深度，%。
double AM_Depth	AM 调制深度，%。
double FM_Deviation	FM 调制频偏，单位 Hz。
double* Phase	PM 解调数据起始内存地址。
uint32_t Phase_Len	PM 解调数据长度。
double* Freq	FM 解调数据起始内存地址。
uint32_t Freq_Len	FM 解调数据长度。
double* Amp	AM 解调数据起始内存地址。
uint32_t Amp_Len	AM 解调数据长度。
double* SSB	SSB 解调数据起始内存地址，上边带和下边带均用此参数。
uint32_t SSB_Len	SSB 解调数据长度。

25.29 Demod_SymbolMap_TypeDef

Demod_SymbolMap_TypeDef 详细定义	
float I	x 轴坐标，表示符号在复数平面中的实部。
float Q	y 轴坐标，表示符号在复数平面中的虚部。

25.30 Pulse_Profile_TypeDef

Pulse_Profile_TypeDef 详细定义	
uint32_t ExpPulseNum	期望获取的脉冲数量，范围：[1，500000]。
Unit_TypeDef unit	脉冲数据单位，dBm 或 V。 详见 Unit_TypeDef 枚举的定义。
float* Pulse	脉冲数据的起始内存地址，数据单位取决于 unit，当使用 Pulse_Detect 函数时，需指向 DET 数据，当使用 Pulse_Detect_PM1 函数时，需指向 IQS 数据，且数据单位必须为 V。
uint64_t PulseSize	脉冲数据长度，范围：[10，500000000]。
double TimeResolution_s	脉冲数据时间分辨率，单位：s。
double DetThreshold	脉冲检测门限，单位与数据保持一致。

25.31 PulseInfo_TypeDef

PulseInfo_TypeDef 详细定义	
uint32_t ActPulseNum	实际获取的脉冲数量，按照此数量可从指针变量中获取每个脉冲的的检测结果。
PulseTDParam_TypeDef* PulseTDParam	脉冲检测时域参数的起始内存地址，结构体内包括脉宽、周期、占空比等，单位均为秒。 详见 PulseTDParam_TypeDef 结构体的定义。
PulseAMPParam_TypeDef* PulseAMPParam	脉冲检测幅度参数的起始内存地址，结构体内包括峰值电平、基准电平、峰基比等。 详见 PulseAMPParam_TypeDef 结构体的定义。
PulseEstParam_TypeDef* PulseEstParam	脉冲检测绘图参数的起始内存地址。 详见 PulseEstParam_TypeDef 结构体的定义。
PulseStatsParam_TypeDef PulseStats	脉冲检测统计参数结构体，包括：最小、最大、平均周期等，单位均为秒。 详见 PulseStatsParam_TypeDef 结构体的定义。

25.32 PulseInfoPM1_TypeDef

PulseInfoPM1_TypeDef 详细定义	
uint32_t ActPulseNum	实际获取的脉冲数量，按照此数量可从指针变量中获取每个脉冲的的检测结果。
PulseTDParam_TypeDef* PulseTDParam	脉冲检测时域参数的起始内存地址，结构体内包括脉宽、周期、占空比等，单位均为秒。 详见 PulseTDParam_TypeDef 结构体的定义。

PulseAMPParam_TypeDef* PulseAMPParam	脉冲检测幅度参数的起始内存地址，结构体内包括峰值电平、基准电平、峰基比等。 详见 PulseAMPParam_TypeDef 结构体的定义。
PulseEstParam_TypeDef* PulseEstParam	脉冲检测绘图参数的起始内存地址。 详见 PulseEstParam_TypeDef 结构体的定义。
PulseStatsParam_TypeDef PulseStats	脉冲检测统计参数结构体，包括：最小、最大、平均周期等，单位均为秒。 详见 PulseStatsParam_TypeDef 结构体的定义。
uint8_t* Mod	脉冲调制类型，0：CW，1：LFM。
PulseFreqPhaseParam_TypeDef* PulseFreqPhase	脉冲的频率和相位信息。 详见 PulseFreqPhaseParam_TypeDef 结构体的定义。

25.33 PulseTDParam_TypeDef

PulseTDParam_TypeDef 详细定义	
double RiseTime	上升时间，单位：s。
double RiseEdge	上升沿。
double FallTime	下降时间，单位：s。
double FallEdge	下降沿。
double Width	脉宽，单位：s。
double Period	周期，单位：s。
float DutyCycle	占空比%。

25.34 PulseAMPParam_TypeDef

PulseAMPParam_TypeDef 详细定义	
float TopLevel_dBm	峰值电平，单位：dBm。
float TopLevel_V	峰值电平，单位：V。
float BaseLevel_dBm	基准电平，单位：dBm。
float BaseLevel_V	基准电平，单位：V。
float TopToBaseRatio_dB	峰基比，单位：dB。
float TopToBaseDiff_V	峰基差，单位：V。
float Droop_dB	下垂，单位：dB。
float Droop_V	下垂，单位：V。
float Overshoot_dB	过冲，单位：dB。
float Overshoot_V	过冲，单位：V。
float Ripple_dB	波纹，单位：dB。

float Ripple_V	波纹，单位：V。

25.35 PulseEstParam_TypeDef

PulseEstParam_TypeDef 详细定义	
double Level_10pct_Index[2]	10% 电平值所在的数组下标，0 为上升沿的下标，1 为下降沿的下标。
double Level_50pct_Index[2]	50% 电平值所在的数组下标，0 为上升沿的下标，1 为下降沿的下标。
double Level_90pct_Index[2]	90% 电平值所在的数组下标，0 为上升沿的下标，1 为下降沿的下标。
double Level_95pct_Index[2]	95% 电平值所在的数组下标，0 为上升沿的下标，1 为下降沿的下标。
double Width_25pct_Index	25% 脉宽位置的数组下标。
double Width_75pct_Index	75% 脉宽位置的数组下标。
uint64_t Start_Index	推测出的信号与噪声的数据起始下标。
uint64_t Size	推测出的信号与噪声的数据长度。
float* Noise_dBm	推测出的噪声数据起始内存地址，单位：dBm。
float* Noise_V	推测出的噪声数据起始内存地址，单位：V。
float* Signal_dBm	推测出的信号数据起始内存地址，单位：dBm。
float* Signal_V	推测出的信号数据起始内存地址，单位：V。

25.36 PulseStatsParam_TypeDef

PulseStatsParam_TypeDef 详细定义	
double MinPRI	最小周期，单位：s。
double MaxPRI	最大周期，单位：s。
double MeanPRI	平均周期，单位：s。
double MinPW	最小脉宽，单位：s。
double MaxPW	最大脉宽，单位：s。
double MeanPW	平均脉宽，单位：s。
float PRIDeviationPercent	周期偏差百分比，%。
float PWDeviationPercent	脉宽偏差百分比，%。

25.37 PulseFreqPhaseParam_TypeDef

PulseFreqPhaseParam_TypeDef 详细定义	
double FreqMean	频率均值，单位：Hz。
double FreqErrorRMS	频率误差。

double PhaseMean	相位均值。
double PhaseErrorRMS	相位误差。

25.38 MSCAN_Profile_TypeDef

MSCAN_Profile_TypeDef 详细定义	
double CenterFreq_Hz	中心频率，单位 Hz。
double RefLevel_dBm	参考电平，单位 dBm。
double DwellTime	每个频率点的停留时间。
uint32_t DecimateFactor	时域数据的抽取倍数。
uint32_t FFTSize	FFT 点数。
uint32_t DetectCount	检波次数。
Detector_TypeDef Detector	检波类型。详见 Detector_TypeDef 枚举的定义。
IFAGC_TypedDef IFAGC	是否使能 IFAGC。详见 IFAGC_TypedDef 枚举的定义。
XPPSTrigger_TypedDef XPPSTrigger	是否使能 XPPSTrigger。详见 XPPSTrigger_TypedDef 枚举的定义。
IQPlayBack_TypedDef IQPlayBack	是否使能 IQPlayBack。详见 IQPlayBack_TypedDef 枚举的定义。
Window_TypeDef Window	窗类型。详见 Window_TypeDef 枚举的定义。

25.39 MSCAN_Info_Typedef

MSCAN_Info_Typedef 详细定义	
int32_t SpectrumFrames	频谱中的帧数。
int32_t SpectrumPoints	每帧频谱的点数。
int32_t IQStreamPoints	IQ 数据中的点数。
double CenterFreq_Hz	中心频率，单位 Hz。
double Span_Hz	频率扫宽，单位 Hz。
double IQSampleRate	当前配置下的 IQ 采样率。

25.40 MSCAN_Data_Typedef

MSCAN_Data_Typedef 详细定义	
int64_t RepeatIndex	当前 Element 已重复到第几次。
int32_t ElementIndex	当前是哪一个 Element。
int Status	元素处理的返回状态。

uint32_t SpectrumFrames	频谱实际的帧数。
uint32_t SpectrumPoints	每帧频谱的点数。
uint32_t IQStreamPoints	实际 IQ 数据的点数。
float ScaleTodBm	由线性功率转对数功率导致的压缩。绝对功率 = SpectrumStream[] * ScaleTodBm + OffsetTodBm。
float OffsetTodBm	相对功率转换成绝对功率的偏移。
float ScaleToV	int16 类型 IQ 至电压绝对值（V）的系数。
uint8_t* SpectrumStream	频谱数据（用户需要分配空间，空间大小为帧数 × 点数）。
int16_t* IQStream	IQ 数据（用户需要分配空间，空间大小为 IQ 数据点数）。
double Temperature	设备温度，摄氏度 = 0.01 * Temperature。
double SysTimeStamp	当前数据包所对应的系统时间戳，单位 s。
double AbsoluteTimeStamp	当前数据包对应的绝对时间戳。
double Latitude	当前数据包对应的纬度坐标，北纬为正数，南纬为负数。
double Longitude	当前数据包对应的经度坐标，东经为正数，西经为负数。
double Altitude	当前数据包对应的海拔，单位米。
double SATHealth	当前 GNSS 定位卫星的健康度（SNR）。
double RefClkFreqOffset	当前设备参考时钟频率的偏移，该值以 GNSS 频率为参考，单位 ppm。
uint64_t nsSinceEpoch	当前数据包所对应的 ns 级系统时间戳。

25.41 AM_DemodParam_TypeDef

AM_DemodParam_TypeDef 详细定义	
float* DemodWaveform	解调后的波形数组，单位：%，长度由 DemodWaveformSize 指定。
float* AFSpectrum_ModDepth	音频谱幅度数组，长度 = DemodWaveformSize / 2，线性比例，单位：%。
double* AFSpectrum_Freq	音频谱对应的频率数组，单位：Hz。
uint32_t DemodWaveformSize	解调波形的采样点数。
float ModDepth	调制深度平均值，单位：%。
float ModDepthPeakPos	正峰值调制深度（Peak+），单位：%。
float ModDepthPeakNeg	负峰值调制深度平均（Peak−），单位：%。
float ModDepthHalfPeak	半峰值调制深度平均值（(Peak+ − Peak−) / 2），单位：%。
float ModDepthRMS	RMS 调制深度平均值，单位：%。
float CarrierPower	载波功率，单位：dBm。
double ModRate	调制速率，单位：Hz。
float SINAD	信号与噪声及失真比，单位：dB。
float RMSPower	RMS 功率，单位：dBm。

double FreqError	频率误差，单位：Hz。
float SNR	信噪比，单位：dB。
float DistTotalVrms	总失真比（RMS），单位：%。
float THD	总谐波失真，单位：%。
float PEP	峰值包络功率（PEP），单位：dBm。

25.42 FM_DemodParam_TypeDef

FM_DemodParam_TypeDef 详细定义	
float* DemodWaveform	解调后的波形数组，单位：Hz，长度由 DemodWaveformSize 指定。
float* AFSpectrum_Deviation	音频谱幅度数组，长度= DemodWaveformSize / 2，线性比例，单位：Hz。
double* AFSpectrum_Freq	音频谱对应的频率数组，单位：Hz。
uint32_t DemodWaveformSize	解调波形的采样点数。
float Deviation	频率偏移量（Deviation），单位：Hz。
float DeviationPeakPos	正峰值频率偏移（Peak+），单位：Hz。
float DeviationPeakNeg	负峰值频率偏移平均（Peak−），单位：Hz。
float DeviationHalfPeak	半峰值频率偏移平均 ((Peak+ − Peak−)/2)，单位：Hz。
float DeviationRMS	RMS 频率偏移平均值，单位：Hz。
float CarrierPower	载波功率，单位：dBm。
double CarrierFreqErr	载波频率误差，单位：Hz。
double ModRate	调制速率，单位：Hz。
float SINAD	信号与噪声及失真比，单位：dB。
float SNR	信噪比，单位：dB。
float DistTotalVrms	总失真比（RMS），单位：%。
float THD	总谐波失真，单位：%。

25.43 ASG_Profile_TypeDef

ASG_Profile_TypeDef 详细定义	
double CenterFreq_Hz	点频模式（SIG_Fixed）下的中心频率，单位：Hz。 输入范围 1 MHz ~ 1 GHz，步进 1 Hz。
double Level_dBm	点频模式（SIG_Fixed）下的输出功率，单位：dBm。 输入范围 -50 ~ 0 dBm，步进 0.25 dB。
double StartFreq_Hz	频率扫描模式（SIG_FreqSweep_*）下的起始频率，单位：Hz。 输入范围 1 MHz ~ 1 GHz，步进 1 Hz。
double StopFreq_Hz	频率扫描模式（SIG_FreqSweep_*）下的终止频率，单位：Hz。 输入范围 1 MHz ~ 1 GHz，步进 1 Hz。

double StepFreq_Hz	频率扫描模式（SIG_FreqSweep_*）下的频率步进，单位：Hz。 输入范围 1 Hz ~ 1 GHz，步进 1 Hz。
double StartLevel_dBm	功率扫描模式下的起始功率，单位：dBm。
double StopLevel_dBm	功率扫描模式下的终止功率，单位：dBm。
double StepLevel_dBm	功率扫描模式下的功率步进，单位：dBm。
double DwellTime_s	频率扫描或功率扫描模式下的驻留时间，单位：s。 当触发源为 BUS 时生效，输入范围 0 ~ 1000000，步进 1。
double ReferenceClockFrequency	参考时钟频率设置，对内参考与外参考均生效。
ReferenceClockSource_TypeDef ReferenceClockSource	参考时钟源选择，用于指定使用内参考或外参考时钟。 详见 ReferenceClockSource_TypeDef 枚举的定义。
ASG_Port_TypeDef Port	模拟信号源的输出端口选择。 详见 ASG_Port_TypeDef 枚举的定义。
ASG_Mode_TypeDef Mode	信号源工作模式选择。 详见 ASG_Mode_TypeDef 枚举的定义。
ASG_TriggerSource_TypeDef TriggerSource	信号源触发输入源配置，默认使用自由运行。 详见 ASG_TriggerSource_TypeDef 枚举的定义。
ASG_TriggerInMode_TypeDef TriggerInMode	信号源的触发输入模式配置，默认使用单点触发。 详见 ASG_TriggerInMode_TypeDef 枚举的定义。
ASG_TriggerOutMode_TypeDef TriggerOutMode	信号源的触发输出模式配置，默认无输出。 详见 ASG_TriggerOutMode_TypeDef 枚举的定义。

25.44 ASG_Info_TypeDef

ASG_Info_TypeDef 详细定义	
uint32_t SweepPoints	扫描点数。

25.45 TraceAnalysisResult_IP3_TypeDef

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	低频三阶互调产物频率，单位随数据源。
double HighIM3PFreq	高频三阶互调产物频率，单位随数据源。
float LowTonePower_dBm	低音信号功率，单位：dBm。
float HighTonePower_dBm	高音信号功率，单位：dBm。
float TonePowerDiff_dB	低音功率与高音功率的差值。
float LowIM3P_dBc	低频三阶互调产物相对于最强主信号的幅度，单位：dBc。

float HighIM3P_dBc	高频三阶互调产物相对于最强主信号的幅度，单位：dBc。
float IP3_dBm	计算得到的三阶互调截点，单位：dBm。

25.46 TraceAnalysisResult_IP2_TypeDef

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	二阶互调产物频率，单位随数据源。
float LowTonePower_dBm	低音信号功率，单位：dBm。
float HighTonePower_dBm	高音信号功率，单位：dBm。
float TonePowerDiff_dB	低音功率与高音功率的差值。
float IM2P_dBc	二阶互调产物相对于最强主信号的幅度，单位：dBc。
float IP2_dBm	计算得到的二阶互调截点，单位：dBm。

25.47 DSP_ChannelPowerInfo_TypeDef

DSP_ChannelPowerInfo_TypeDef 详细定义	
float ChannelPower_dBm	信道内的总功率，单位：dBm。
float PowerDensity	信道功率密度，单位：dBm/Hz。
float ChannelPeakIndex	信道内功率峰值对应的迹线索引。
double ChannelPeakFreq_Hz	信道内功率峰值对应的频率，单位：Hz。
float ChannelPeakPower_dBm	信道内功率峰值，单位：dBm。

25.48 TraceAnalysisResult_XdB_TypeDef

TraceAnalysisResult_XdB_TypeDef 详细定义	
double XdBBandWidth_Hz	XdB 带宽，单位：Hz。
double CenterFreq_Hz	XdB 带宽对应的中心频率，单位：Hz。
double StartFreq_Hz	XdB 带宽的起始频率，单位：Hz。
double StopFreq_Hz	XdB 带宽的终止频率，单位：Hz。
float StartPower_dBm	XdB 带宽起始频率对应的功率，单位：dBm。
float StopPower_dBm	XdB 带宽终止频率对应的功率，单位：dBm。
uint32_t PeakIndex	XdB 带宽范围内的峰值所在的迹线索引。
double PeakFreq_Hz	XdB 带宽范围内的峰值频率，单位：Hz。
float PeakPower_dBm	XdB 带宽范围内的峰值功率，单位：dBm。

25.49 TraceAnalysisResult_OBW_TypeDef

TraceAnalysisResult_OBW_TypeDef 详细定义	
double OccupiedBandWidth	占用带宽，单位：Hz。
double CenterFreq_Hz	占用带宽的中心频率，单位：Hz。
double StartFreq_Hz	占用带宽的起始频率，单位：Hz。
double StopFreq_Hz	占用带宽的终止频率，单位：Hz。
float StartPower_dBm	占用带宽起始频率对应的功率，单位：dBm。
float StopPower_dBm	占用带宽终止频率对应的功率，单位：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，单位：Hz。
float PeakPower_dBm	占用带宽内的峰值功率，单位：dBm。

25.50 DSP_ACPRFreqInfo_TypeDef

DSP_ACPRFreqInfo_TypeDef 详细定义	
double RBW	分析使用的分辨率带宽，单位：Hz。
double MainChCenterFreq_Hz	主信道中心频率，单位：Hz。
double MainChBW_Hz	主信道带宽，单位：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，单位：Hz。
uint32_t AdjChPair	邻道对数量，1 表示左右各 1 个邻道，2 表示左右各 2 个邻道。

25.51 TraceAnalysisResult_ACPR_TypeDef

TraceAnalysisResult_ACPR_TypeDef 详细定义	
float MainChPower_dBm	主信道总功率，单位：dBm。
uint32_t MainChPeakIndex	主信道峰值对应的迹线索引。
double MainChPeakFreq_Hz	主信道峰值频率，单位：Hz。
float MainChPeakPower_dBm	主信道峰值功率，单位：dBm。
double L_AdjChCenterFreq_Hz	左邻道中心频率，单位：Hz。
double L_AdjChBW_Hz	左邻带宽，单位：Hz。
float L_AdjChPower_dBm	左邻道功率，单位：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），单位：dBc。
float L_AdjChPeakIndex	左邻道峰值对应的迹线索引。
double L_AdjChPeakFreq_Hz	左邻道峰值频率，单位：Hz。
float L_AdjChPeakPower_dBm	左邻道峰值功率，单位：dBm。
double R_AdjChCenterFreq_Hz	右邻道中心频率，单位：Hz。
double R_AdjChBW_Hz	右邻道带宽，单位：Hz。
float R_AdjChPower_dBm	右邻道功率，单位：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），单位：dBc。
float R_AdjChPeakIndex	右邻道峰值对应的迹线索引。
double R_AdjChPeakFreq_Hz	右邻道峰值频率，单位：Hz。
float R_AdjChPeakPower_dBm	右邻道峰值功率，单位：dBm。

25.52 DSP_FFT_TypeDef

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	指定 FFT 分析所使用的窗函数，默认使用 FlatTop 窗。 详见 Window_TypeDef 枚举的定义。
TraceDetector_TypeDef TraceDetector	设置迹线检波器类型，默认使用正峰值检波。 详见 TraceDetector_TypeDef 枚举的定义。
uint32_t DetectionRatio	迹线检波比。
float Intercept	输出频谱截取比例，例如 Intercept = 0.8 表示输出 80%的频谱结果。
bool Calibration	设置是否进行校准，0：不进行，1：进行。

25.53 DSP_DDC_TypeDef

DSP_DDC_TypeDef 详细定义	
double DDCOffsetFrequency	数字下变频（DDC）过程中使用的复混频频率偏移，用于将目标信号从原始中心频率搬移至基带。
double SampleRate	DDC 输出数据的采样率，用于确定下变频及后续处理的数据速率。
float DecimateFactor	重采样抽取倍数，用于降低数据速率与带宽，取值范围为 1 ~ 2^{16}。
uint64_t SamplePoints	DDC 输出的有效采样点数，决定一次下变频处理所生成的数据长度。

25.54 DSP_AudioAnalysis_TypeDef

DSP_AudioAnalysis_TypeDef 详细定义	
double AudioVoltage	音频信号的有效电压值，单位：V。
double AudioFrequency	音频信号的基波频率，单位：Hz。
double SINDA	音频信号的信纳德，单位：dB。
double THD	音频信号的总谐波失真，单位：%。

25.55 Filter_TypeDef

Filter_TypeDef 详细定义	
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，单位：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。

25.56 DeviceFirmwareVersion_TypeDef

DeviceFrimwareVersion_TypeDef 详细定义	
uint32_t FFWVersion	FPGA 固件版本。
uint32_t MFWVersion	MCU 固件版本。
uint32_t BusVersion	总线固件版本。
uint16_t PMUVersion	PMU 固件版本。
uint16_t AGUVersion	AGU 固件版本。

26. 枚举变量

26.1 PhysicalInterface_TypeDef

USB	使用 USB 作为物理接口，适用于 USB 型设备；
ETH	使用 100M/1000M 以太网作为物理接口，适用于网口型设备。

26.2 DevicePowerSupply_TypeDef

USBPortAndPowerPort	通过 USB 数据端口和电源端口同时供电；

26.3 IPVersion_TypeDef

IPv4	使用 IPv4 地址；

26.4 SysPowerState_TypeDef

PowerON	系统所有工作区均上电；
RFPowerOFF	射频下电，不可快速唤醒；
RFStandby	射频待机，可快速唤醒。

26.5 SysPowerMode_TypeDef

SysPowerMode_Normal	MCU 上电模式；
SysPowerMode_Standby	MCU 待机模式；
SysPowerMode_Stop	MCU 掉电模式。

26.6 FanState_TypeDef

FanState_On	强制常开；
FanState_Off	强制常关；
FanState_Auto	自动模式，设备温度 ≥ 50℃ 自动开启，降至 ≤ 40℃ 自动关闭。

26.7 ClkCalibrationSource_TypeDef

CalibrateByExternal	通过外部触发信号校准时钟；
CalibrateByGNSS1PPS	通过 GNSS 1PPS 信号校准时钟。

26.8 GNSSPeriphType_TypeDef

GNSS_None	无外设；
GNSS_For_EIO	EIO 外设；
GNSS_For_NX	NX 外设；
GNSS_For_PX	PX 外设；
GNSS_For_PXZ	PXZ 外设。

GNSS_For_TG	TG 外设

26.9 GNSSType_TypeDef

None_GPS	无 GPS 接收机;
GNSS_GPS	标准 GPS;
GNSS_GPS_Pro	高性能 GPS。

26.10 OCXOType_TypeDef

None_OCXO	无 OCXO;
GNSS_OCXO	普通 OCXO;
GNSS_DOCXO	可调谐 OCXO。

26.11 GNSSAntennaState_TypeDef

GNSS_AntennaExternal	外部天线;
GNSS_AntennaInternal	内部天线。

26.12 DOCXOWorkMode_TypeDef

DOCXO_LockMode	驯服模式;
DOCXO_HoldMode	跟踪模式。

26.13 SWP_FreqAssignment_TypeDef

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。

26.14 Window_TypeDef

FlatTop	具有良好的幅度准确度;
Blackman_Nuttall	具有良好的频率分辨力;
LowSideLobe	具有良好的抑制频谱泄漏能力;
Rect	矩形窗，等效于不加窗，直接对有限长度信号进行截断;
Kaiser	凯泽窗，主瓣宽度与旁瓣抑制可按需求配置，适合高动态范围测量。

26.15 RBWMode_TypeDef

RBW_Manual	手动输入 RBW;
RBW_Auto	自动随 SPAN 更新 RBW;
RBW_OneThousandthSpan	强制 RBW = 0.001 * SPAN;
RBW_OnePercentSpan	强制 RBW = 0.01*SPAN。

26.16 VBWMode_TypeDef

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
VBW_TenTimesRBW	强制 VBW = 10 * RBW，完全旁路 VBW 滤波器。

26.17 SweepTimeMode_TypeDef

SWTMode_minSWT	以最短扫描时间进行扫描;
SWTMode_minSWTx2	以近似 2 倍最短扫描时间进行扫描;
SWTMode_minSWTx4	以近似 4 倍最短扫描时间进行扫描;
SWTMode_minSWTx10	以近似 10 倍最短扫描时间进行扫描;
SWTMode_minSWTx20	以近似 20 倍最短扫描时间进行扫描;
SWTMode_minSWTx50	以近似 50 倍最短扫描时间进行扫描;
SWTMode_minSWTxN	以近似 N 倍最短扫描时间进行扫描，N 等于 SweepTimeMultiple;
SWTMode_Manual	以近似指定的扫描时间进行扫描，扫描时间等于 SweepTime。
SWTMode_minSMPxN	以近似 N 倍最短采样时间进行单个频点的采样，N 等于 SampleTimeMultiple

26.18 Detector_TypeDef

Detector_Sample	每个频点的功率谱间不进行帧间检波;
Detector_PosPeak	每个频点的功率谱间进行帧检波，最终输出一帧，帧与帧取 MaxHold;
Detector_Average	每个频点的功率谱间进行帧检波，最终输出一帧，帧与帧取平均;
Detector_NegPeak	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;
Detector_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）;
Detector_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）;
Detector_RMS	每个频点的功率谱间进行帧检波，最终输出一帧，帧与帧取 RMS。

26.19 TraceFormat_TypeDef

TraceFormat_Standard	频率等间隔映射，适用于常规频谱观测和快速扫描;
TraceFormat_PrecisFrq	频率准确映射，适用于对信号频率读数有严苛要求的测量。

26.20 TraceDetectMode_TypeDef

TraceDetectMode_Auto	自动选择迹线检波模式;
TraceDetectMode_Manual	指定迹线检波模式。

26.21 TraceDetector_TypeDef

TraceDetector_AutoSample	自动取样检波；
TraceDetector_Sample	取样检波；
TraceDetector_PosPeak	正峰值检波；
TraceDetector_NegPeak	负峰值检波；
TraceDetector_RMS	均方根检波；
TraceDetector_Bypass	不执行检波；
TraceDetector_AutoPeak	自动峰值检波；
TraceDetector_Normal	正态检波。

26.22 TracePointsStrategy_TypeDef

SweepSpeedPreferred	优先保证扫描速度最快，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。
BinSizeAssined	优先保证按照设定的迹线点数来生成迹线。 此映射策略下，返回的迹线点数等于实际下发的迹线点数，频点间距=（终止频率-起始频率）/（迹线点数-1），可通过这种策略控制返回迹线的两点间间隔（TraceBinBW_Hz），但扫速较慢。

26.23 TraceAlign_TypeDef

NativeAlign	自然对齐，返回比下发频段稍宽的频谱数据。
AlignToStart	对齐至起始频率，精确对齐频段的起始频率，确保频谱数据从起始频率开始。

26.24 FFTExecutionStrategy_TypeDef

Auto	根据设置自动选择使用 CPU 还是 FPGA 进行 FFT 计算。 RBW<40 kHz 是在 CPU 上做处理，RBW≥40 kHz 是在 FPGA 上做处理，具体处理包括：VBW、检波、杂散抑制、FFT 和迹线检波。
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，RBW 较小时，扫速会减慢。

26.25 RxPort_TypeDef

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。仅可选配内置信号源设备适用。

26.26 SpurRejection_TypeDef

Bypass	不进行杂散抑制
Standard	标准杂散抑制
Enhanced	增强杂散抑制

26.27 ReferenceClockSource_TypeDef

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，选配高品质 GNSS 模块后可用。
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。
ReferenceClockSource_External_SysClock	外部系统时钟，直接将外部的时钟输入作为系统时钟使用，旁路内部的参考锁相环。

26.28 SystemClockSource_TypeDef

SystemClockSource_Internal	使用内部系统时钟源；
SystemClockSource_External	使用外部系统时钟源，需将外部系统时钟频率设置为10MHz。

26.29 SWP_TriggerSource_TypeDef

InternalFreeRun	内部触发自由运行；
ExternalPerHop	外部触发，每一次触发都跳一个频点；
ExternalPerSweep	外部触发，每一次触发都刷新一条迹线；
InternelXppsPerHop	内部 GNSS 秒脉冲触发，每一次触发都跳一个频点；
InternelXppsPerSweep	内部 GNSS 秒脉冲触发，每一次触发都刷新一条迹线；
InternelXppsPerProfile	内部 GNSS 秒脉冲触发，每一次触发都应用一个新配置。

26.30 TriggerEdge_TypeDef

RisingEdge	由上升沿触发。
FallingEdge	由下降沿触发。

26.31 TriggerOutMode_TypeDef

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

26.32 TriggerOutPulsePolarity_TypeDef

Positive	正脉冲
Negative	负脉冲

26.33 GainStrategy_TypeDef

LowNoisePreferred	侧重低噪声，设置后前置放大器为自动使能状态，参考电平降低至-30 dBm 左右，前放开启。
HighLinearityPreferred	侧重高线性度，设置后强制关闭前置放大器。

26.34 PreamplifierState_TypeDef

AutoOn	自动使能前置放大器，参考电平降低至-30 dBm 左右，前放开启；
ForcedOff	强制保持前置放大器关闭；
OnLowGain	放大器低增益模式；
OnMediumGai	放大器中增益模式；
OnHighGain	放大器高增益模式。

26.35 SWP_TraceType_TypeDef

ClearWrite	输出正常迹线。
MaxHold	输出经过最大值保持的迹线。
MinHold	输出经过最小值保持的迹线。
ClearWriteWithIQ	同时输出当前频点的时域数据与频域数据。

26.36 LOOptimization_TypeDef

LOOpt_Auto	本振优化，自动
LOOpt_Speed	本振优化，高扫速
LOOpt_Spur	本振优化，低杂散
LOOpt_PhaseNoise	本振优化，低相噪

26.37 DSPPlatform_Typedef

CPU_DSP	在 CPU 进行计算
FPGA_DSP	在 FPGA 进行计算

26.38 SWPApplication_TypeDef

SWPNoiseMeas	显示平均噪声电平测量
SWPChannelPowerMeas	信道功率测量
SWPOBWMeas	占用带宽测量
SWPACPRMeas	邻道功率比测量

SWPIM3Meas	IP3/IM3 测量

26.39 RxPort_TypeDef

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

26.40 IQS_TriggerSource_TypeDef

External	外部触发，由连接至设备外触发输入端口的物理信号进行触发。
Bus	总线触发。由函数（指令）的方式进行触发。
Level	电平触发。设备根据设定的电平门限对输入信号进行检测，当输入超过门限值后自动发起触发；
Timer	定时器触发。使用设备内部定时器以设定的时间周期进行自动触发；
TxSweep	由设备内部信号源扫描触发数据采集（需 ASG 选件）当选择此触发源时，采集过程将由信号源扫描的输出触发信号进行触发；
MultiDevSyncByExt	在外部触发信号到来时，多机做同步触发行为；
MultiDevSyncByGNSS1PPS	设备搭配的 GNSS 模块输出的 GNSS-1PPS 信号到来时，多机做同步触发行为；
GNSS1PPS	使用系统内 GNSS 提供的 1PPS 进行触发，需搭配 GNSS 模块。

26.41 TriggerMode_TypeDef

FixedPoints	触发后，采集定点长度（TriggerLength）的数据，采集期间不响应触发，采集完成后设备重新进入待触发状态；
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）。

26.42 TriggerTimerSync_TypeDef

NoneSync	定时器触发不与外触发同步；
SyncToExt_RisingEdge	定时器触发与外触发上升沿同步；
SyncToExt_FallingEdge	定时器触发与外触发下降沿同步；
SyncToExt_SingleRisingEdge	定时器触发与外触发上升沿单次同步（需要调用指令函数，执行单次同步）；
SyncToExt_SingleFallingEdge	定时器触发与外触发下降沿单次同步（需要调用指令函数，执行单次同步）；
SyncToGNSS1PPS_RisingEdge	定时器触发与 GNSS-1PPS 上升沿同步（需搭配 GNSS 模块）；
SyncToGNSS1PPS_FallingEdge	定时器触发与 GNSS-1PPS 下降沿同步（需搭配 GNSS 模块）；
SyncToGNSS1PPS_ SingleRisingEdge	定时器触发与 GNSS-1PPS 上升沿单次同步（需要调用指令函数，执行单次同步，需搭配 GNSS 模块）；
SyncToGNSS1PPS_ SingleFallingEdge	定时器触发与 GNSS-1PPS 下降沿单次同步（需要调用指令函数，执行单次同步，需搭配 GNSS 模块）。

26.43 DataFormat_TypeDef

Complex16bit	IQ 两路数据为 16 位格式;
Complex32bit	IQ 两路数据为 32 位格式;
Complex8bit	IQ 两路数据为 8 位格式;
Real16bit	Real，单路数据 16 位;
Real32bit	Real，单路数据 32 位;
Real8bit	Real，单路数据 8 位;
Realfloat	Real，单路数据 32 位 float。

26.44 DCCancelerMode_TypeDef

DCCOff	关闭直流抑制
DCCHighPassFilterMode	开启高通滤波器模式（更好的直流抑制效果，但会抑制 DC 至 100kHz 范围内的信号分量）
DCCManualOffsetMode	开启手动偏置模式，该模式下需人工手动配置 DCCIOffse 和 DCCQOffset 的偏置值，且抑制效果弱于高通滤波器模式，但不会抑制直流上的信号分量;
DCCAutoOffsetMode	开启自动偏置方式，该模式下自动配置 DCCIOffse 和 DCCQOffset 的偏置值

26.45 QDCMode_TypeDef

QDCOff	关闭 QDC 功能
QDCManualMode	开启并使用手动模式；该模式下根据用户手动设置的 QDCIGain、QDCQGain、QDCPhaseComp 进行配置
QDCAutoMode	开启并使用自动 QDC 模式。该模式会自动配置 QDCIGain、QDCQGain、QDCPhaseComp 为默认值

26.46 RBWFilterType_TypeDef

RBWFilter_80PercentABW	最大带宽模式，提供 80%采样率的可用带宽。
RBWFilter_Gaussian_3dB	标准高斯滤波器，以-3 dB 功率点定义带宽。具有优秀的瞬态响应特性（无过冲），是频谱分析中最通用的默认滤波器。
RBWFilter_Gaussian_6dB	以-6 dB 幅度点定义带宽的高斯滤波器。主要用于符合 CISPR 标准的电磁兼容（EMC/EMI）测试。
RBWFilter_Gaussian_Impulse	高斯脉冲滤波器，用于真实还原脉冲信号的时域波形。
RBWFilter_Gaussian_Noise	经过等效噪声带宽校准的滤波器，用于精确测量噪声功率谱密度。
RBWFilter_Flattop	平顶滤波器，用于消除频率微小偏移带来的幅值误差

26.47 LookBack_TypeDef

LookBack_Off	关闭 lookback 功能，不上传原始的 IQ 数据；
LookBack_On	开启 lookback 功能，上传原始的 IQ 数据。

26.48 IFAGC_TypedDef

IFAGC_Off	中频自动增益控制关闭；
IFAGC_On	中频自动增益控制打开。

26.49 XPPSTrigger_TypedDef

XPPSTrigger_Off	关闭 XPPSTrigger；
XPPSTrigger_On	使能 XPPSTrigger。

26.50 IQPlayBack_TypedDef

IQPlayBack_Off	关闭 IQPlayBack；
IQPlayBack_On	使能 IQPlayBack。

26.51 ASG_Port_TypeDef

ASG_Port_External	外部端口；
ASG_Port_Internal	内部端口。

26.52 ASG_Mode_TypeDef

ASG_Mute	静音；
ASG_FixedPoint	定点；
ASG_FrequencySweep	频率扫描；
ASG_PowerSweep	功率扫描；
ASG_TrackGenerator	跟踪信号发生器功能。

26.53 ASG_TriggerSource_TypeDef

ASG_TriggerIn_FreeRun	自由运行；
ASG_TriggerIn_External	外触发；
ASG_TriggerIn_Bus	定时器触发。

26.54 ASG_TriggerInMode_TypeDef

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）。

26.55 ASG_TriggerOutMode_TypeDef

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）。

26.56 Demod_FilterType_TypeDef

RootRaisedCosine	根升余弦滤波器。

26.57 Unit_TypeDef

Voltage_V	电压，V;
Power_dBm	功率，dBm。

附录 1：API 返回值索引

表格 8 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。
-49	当前设备固件版本与 API 版本非相同基线版本	警告	参考第三章基线版本对应表，调整更新当前设备的固件版本或 API 版本。
10054	设备网络连接断开	错误	检查网络配置，确认无误后重新连接。
10060	连接尝试失败，目标主机无响应	错误	检查网络配置，确认无误后重新连接。
10062	设备未正常获取到数据	错误	检查网络配置，确认无误后重新连接。
-50	脉冲检测许可证文件缺失	错误	请将 xx_pulsedet.lic 放到/CalFile/文件夹中。

-51	脉冲检测许可证文件内容错误	错误	通常为许可证内容被改动，请联系技术支持人员。
-52	脉冲检测许可证文件过期	错误	请联系技术支持人员。
-53	脉冲数量错误	错误	期望获取的脉冲数量为 0，不执行脉冲检测，请调整脉冲数量。
-54	脉冲检测数据点数小于下限	错误	不执行脉冲检测，请调整脉冲检测数据点数。
50	脉冲数量超限	警告	期望获取的脉冲数量大于上限，按照上限执行。
51	脉冲检测数据点数超限	警告	脉冲检测数据点数大于上限，按照上限执行。
-60	解调许可证文件缺失	错误	请将 xx_demodlic.txt 文件放到/CalFile/文件夹中。
-61	解调许可证文件内容错误	错误	通常为许可证内容被改动，请联技系术支持人员。
-62	缺少解调库	错误	请将 DigitalSigDemod.dll(Windwos)或 libDigitalSigDemod.so(Linux)放到 htra_api 库的同路径下。
-63	打开解调功能失败	错误	通常不会出现此错误，若出现请联系技术支持人员。
-64	输入的采样点数小于 16384	错误	请将采样点改为 16384 及以上。
-65	输入的采样点数大于 320000	错误	请将采样点数改为 320000 及以下。
-66	输入的采样率/符号率小于 4	错误	请将采样率/符号率改为 4 及以上。
-67	输入的采样率/符号率大于 64	错误	请将采样率/符号率改为 64 及以下。
-68	输入的符号率小于等于 0	错误	请将符号率改为 0 以上。
-69	无解调数据输出	错误	当输入的 IQ 数据存在大量的 0 值时，则可能出现无法解调的情况。
-70	解调失败	错误	检查相关参数配置是否合理。
66	采样点数超限	警告	建议采样点数在满足上下限的前提下，设置为大于等于 2000*SampleRate/SymbolRate。
67	滤波器系数超限	警告	滤波器系数超出可设范围[0.01, 0.99]，将默认使用 0.35。

[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。

附录 2：RTA 概率密度图 BitMap 绘制指南

绘制频谱迹线

在 RTA 模式下，用户可通过 **RTA_GetRealTimeSpectrum** 或 **RTA_GetRealTimeSpectrum_Raw** 函数中返回的 **SpectrumTrace[]**数组与 **RTA_Configuration** 函数返回的 **FrameInfo** 结构体进行频谱迹线的绘制。

1. 频谱数据结构说明

SpectrumStream[]是由多帧频谱的功率数据拼接而成的集合，用户需结合 **FrameInfo** 结构体中的参数对数组进行拆解：

- 单帧宽度（FrameInfo.FrameWidth）：表示每条迹线包含的功率点数；
- 总帧数：（FrameInfo.PacketFrame）：本次返回包含的迹线条数；
- 总有效点数（FrameInfo.PacketValidPoints）：SpectrumStream[]数组的总长度，计算公式为：

  FrameInfo.PacketValidPoints = FrameInfo.PacketFrame * FrameInfo.FrameWidth

2. 绘制频率轴

RTA 模式下只返回功率轴的数据，频率轴需按返回的起始终止频率，以及计算出的频点间隔，自行绘制，绘制方法如下：

- 起始频率：FrameInfo.StartFrequency_Hz；
- 终止频率：FrameInfo.StopFrequency_Hz；
- 每帧点数：FrameInfo.FrameWidth；

频率轴范围：FrequencyRange = FrameInfo.StopFrequency_Hz - FrameInfo.StartFrequency_Hz

频点间隔：FrequencyStep = FrequencyRange / (FrameInfo.FrameWidth - 1)

3. 绘制功率轴

返回的 SpectrumStream[]数组是相对功率，可通过以下方式转换为绝对功率，共有 FrameInfo.PacketFrame 帧数据，仅需取出其中一帧进行绘制。

- 相对功率：SpectrumStream[]；
- 绝对功率：SpectrumStream[] * PlotInfo.ScaleTodBm + PlotInfo.OffsetTodBm

绘制频谱迹线时，可以在 UI 中只取第一帧的数据进行显示，即只取绝对功率数组中前 FrameInfo.FrameWidth 个点进行显示

绘制概率密度图 BitMap

概率密度图通过 **RTA_GetRealTimeSpectrum** 函数中 **SpectrumBitMap[]**数组获取，是一个长度为 **FrameInfo.FrameWidth * FrameInfo.FrameHeight** 的数组。概率密度图阵列的每个元素值都表示概率密度图中每个像素的命中计数。

1．绘图逻辑

将 **SpectrumBitMap[]**中每 **FrameInfo.FrameWidth** 个元素为一行，按照“从左至右，从上至下”的方式连续绘制 **FrameHeight** 行；

2．坐标轴的物理绘制

- X 轴：参照绘制频率轴章节进行绘制；
- Y 轴：从下往上共 FrameHeight 行，y 值范围为 0 至（FrameHeight-1），每个纵向像素位置 y 对应的功率值（单位：dBm）可通过以下方式进行计算：

$$Power_{dBm} = y * PlotInfo.ScaleTodBm + PlotInfo.OffsetTodBm$$

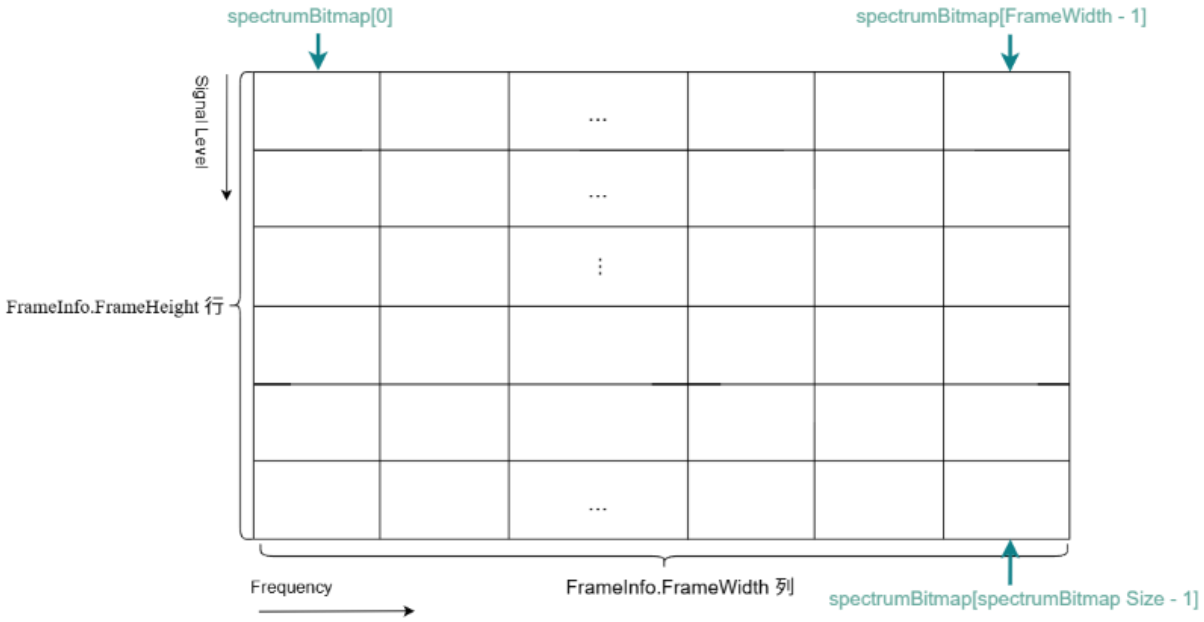


图 11 概率密度图绘制说明

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