API 编程指南

API Version 0.55.62

2025-8-19

HAROGIC
海 得 科 技

目录

1 版本信息

版本	内容	时间
0.54.0	初版 1. 设备与API版本介绍 2. 函数类别简介 3. API使用方法 4. API调用逻辑与调用地图 5. 重要变量及设置概念 6. 系统Device，SWP，IQS，DET，RTA，MPS，ASG，DSP，ASD的相关函数用法及其参数解释。 7. 返回值索引目录。	5-17-2023
0.55.27	1. 修改：重构原本设备与系统章节内容； 2. 新增：SWP模式 SWP_AutoSet 函数。 3. 新增：系统 Device与GNSS相关函数章节。	3-28-2024
0.55.61	1. 新增：相位噪声测量模式（Phase Noise Measurement）章节； 2. 新增：IQS模式 IQS_GetIQStream_PM2 函数； 3. 新增：零扫宽模式（ZeroSpan）章节； 4. 新增：RTA模式RTA_GetRealTimeSpectrum_Raw函数； 5. 新增：数字解调模式（Digital Demod）章节； 6. 新增：脉冲检测（Pulse Det）章节； 7.新增：迹线检波器和检波器的相关说明； 8. 删除：API在Windows/Linux下的使用方法（已在API范例使用指南中说明）； 9. 删除：SWP模式GetPatialUpdatedFullSweep函数； 10. 修改：设备与系统Device章节新增Device_SetNetworkDeviceIP, Device_SetNetworkDeviceIP_PM1, Device_GetFullUID等函数； 11. 修改：函数所有示例全部修改为最新示例。	7-16-2025
0.55.62	1.新增：DSP新增DSP_SEMAnalysis函数；	8-19-2025

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）、IQ数据记录模式（IQS）、检波分析模式（DET）、实时频谱模式（RTA）。充分理解各工作模式的执行机制，并根据应用目标选择合适的工作模式与设备参数，有助于充分发挥设备的工作效能，并获取更为准确的测量结果。

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

API调用的基本流程包括五个步骤：

第一步，打开设备资源；

第二步，根据工作模式调用相应类别的API函数，将设备配置至指定的工作模式；

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

3 设备及 API 版本

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

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

例如：MFM版本 = 0.55.1，FFM版本 = 0.55.2，API版本 = 0.55.4，SAStudio4版本 = 1.55.45，此组合所有软件具有y = 55，则可以匹配运行。

例如：MFM版本 = 0.38.1，FFM版本 = 0.38.4，API版本 = 0.54.2，SAStudio4版本 = 1.55.45，此组合中API版本为y = 54，与MFM、FFM、SAStudio4的版本不匹配，则无法运行或可能得到错误的结果。

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

4 函数类别简介

表1 API 函数类别简介

函数类别	说明
设备与系统 Device	全局功能，可在任意工作模式下调用此类别下的函数。该类别包括对设备打开、关闭、全局性设置、获取设备信息和获取设备状态等功能。
标准频谱分析 SWP	该模式下，**接收机根据配置进行跳频以实现频率扫描**，基带对每个频点获取的分析带宽内的时域数据进行**频谱分析**，并将频谱结果返回给用户。SWP模式适用于面向频率迹线的测量与分析应用。 该类别包括对SWP模式进行配置、获取频谱数据、触发控制等功能。
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。
辅助信号源ASG	全局功能，控制设备或其中的**模拟信号源选件**，可在任意工作模式下调用。该类别包括设置输出单音、扫频等功能。
信号处理DSP	通用后处理函数，与硬件状态无关。 该类别包括对IQ数据进行DDC、FFT分析、视频检波等；对频谱迹线进行测量分析，比如IM3、相位噪声、信道功率、占用带宽等功能。
模拟调制解调 ASD	模拟解调类后处理函数，与硬件状态无关。 该类别包括AM解调、FM解调等功能。

5 API 的调用逻辑与调用地图

5.1 标准扫频分析（SWP）的 API 调用地图

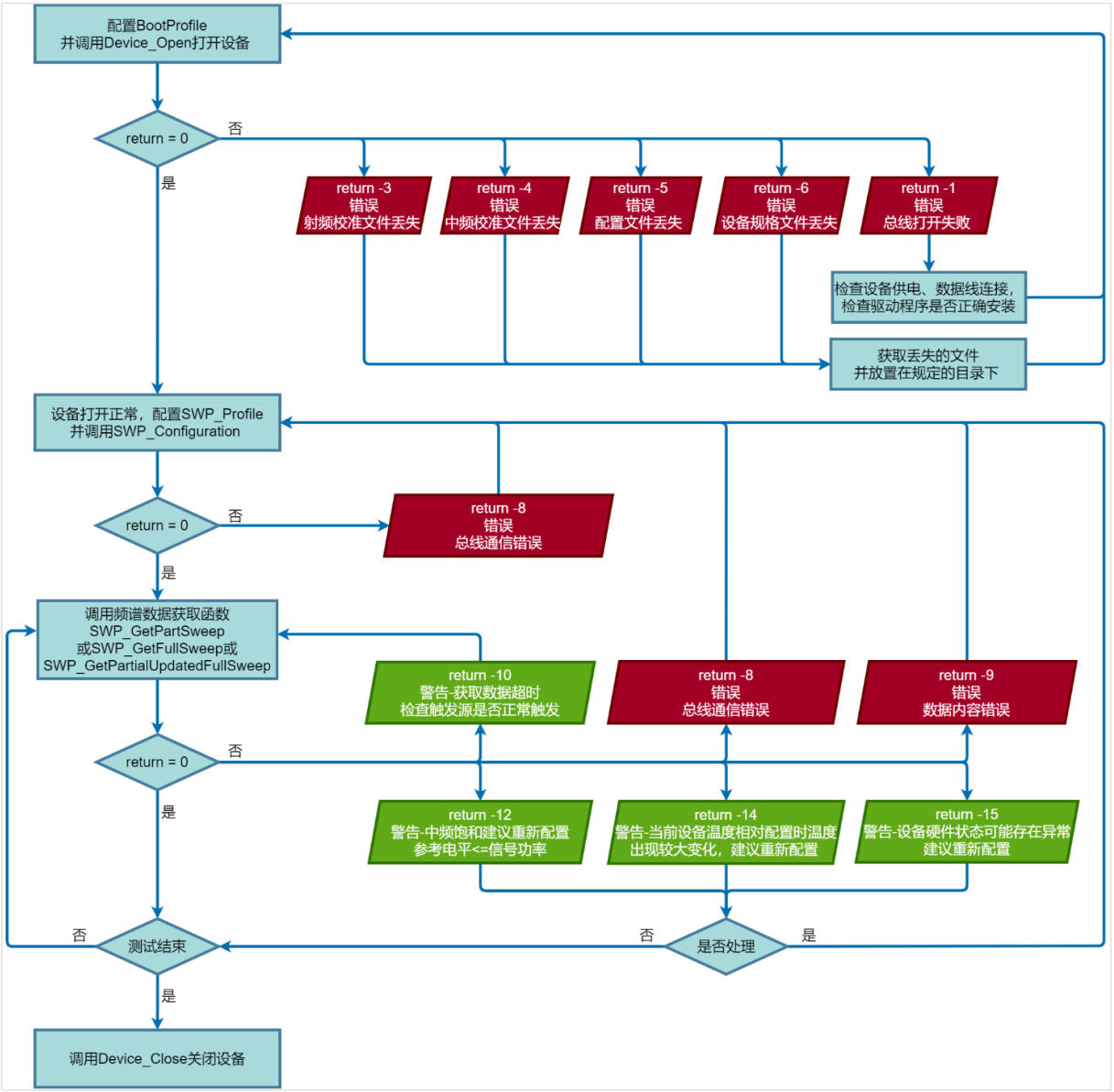

图2　SWP 模式调用流程图

5.2 时域信号记录分析（IQS）的 API 调用地图

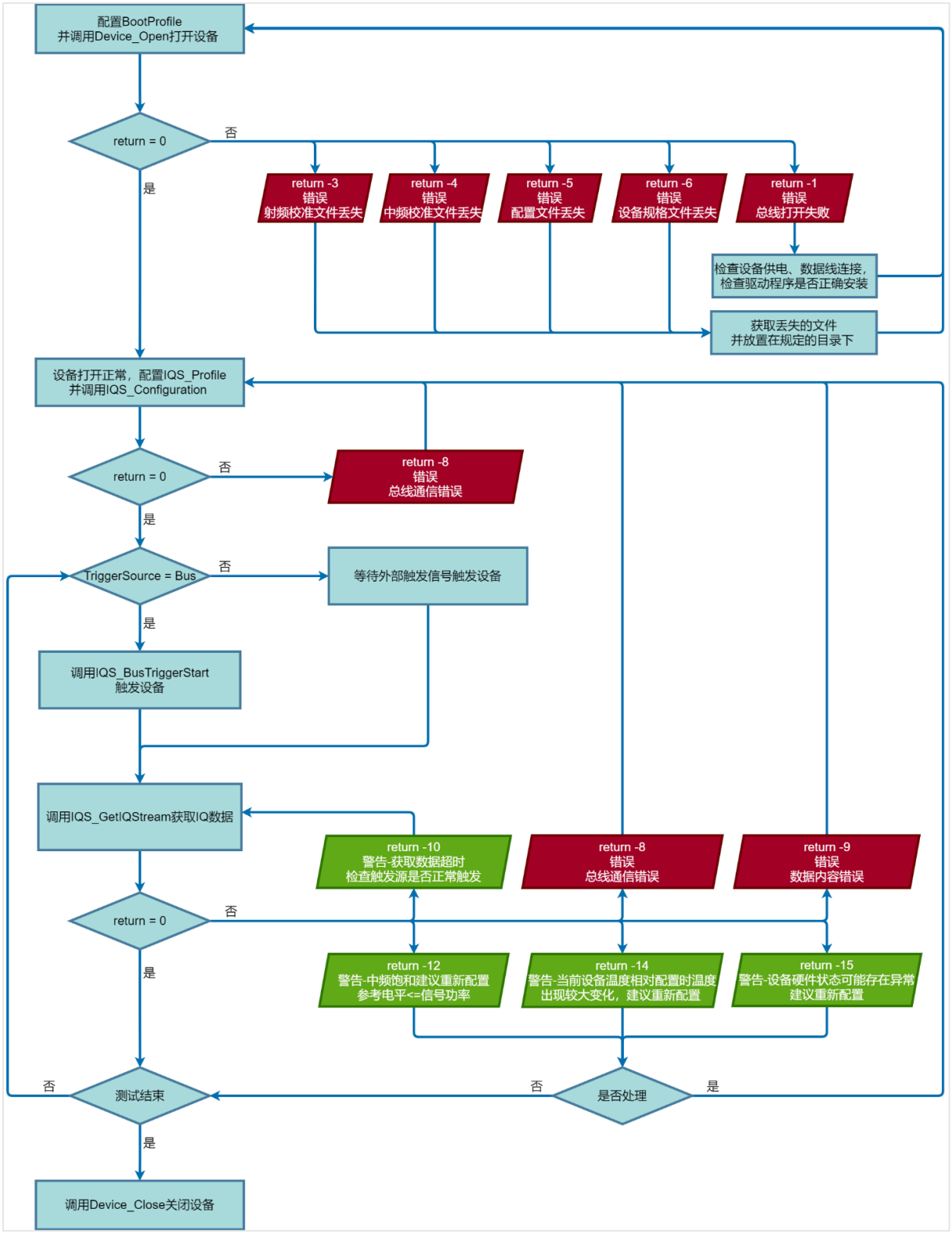

图3　IQS 模式调用流程图(触发模式为 Fixed)

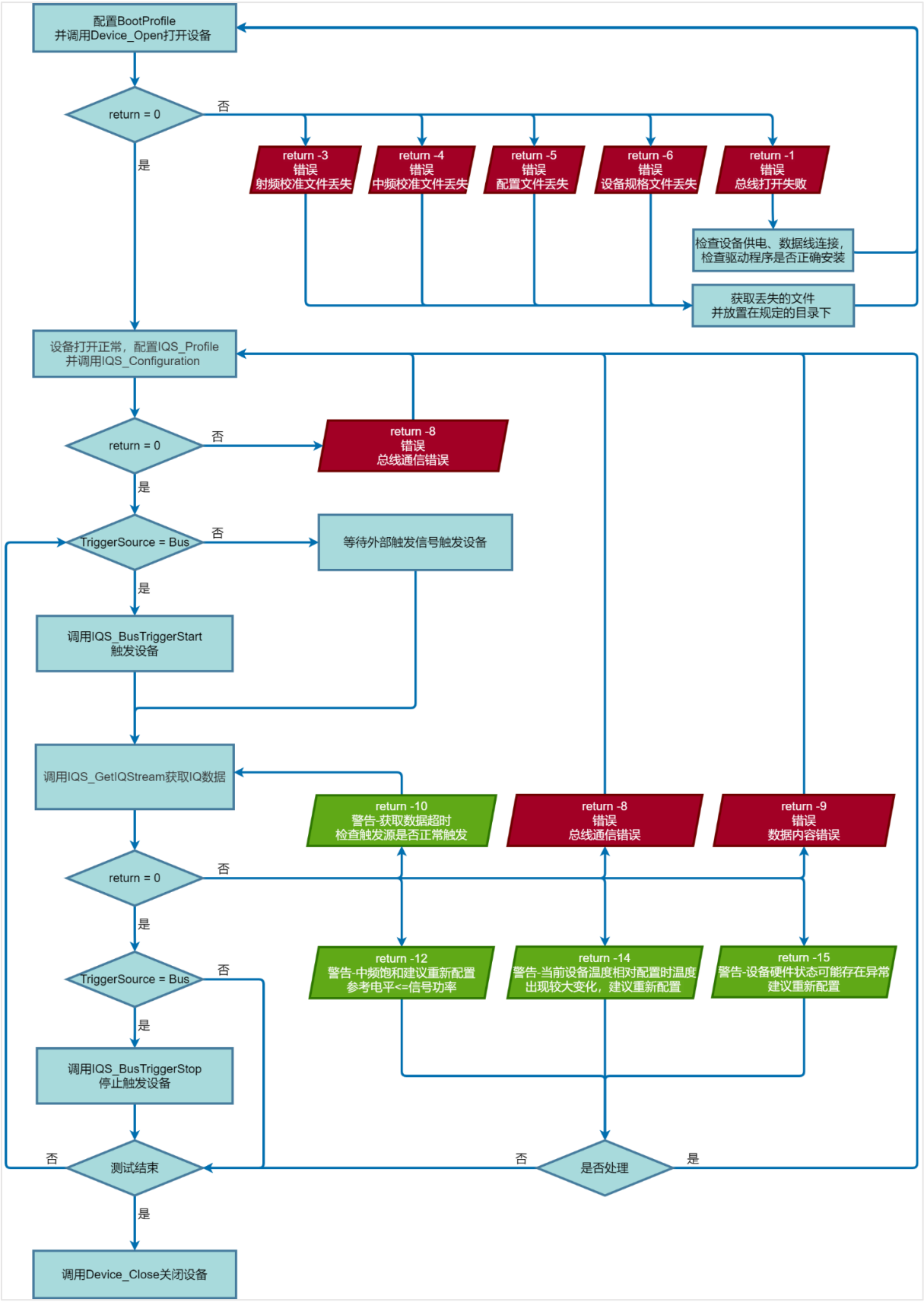

图4　IQS 模式调用流程图(触发模式为 Adaptive)

5.3 检波分析（DET）的 API 调用地图

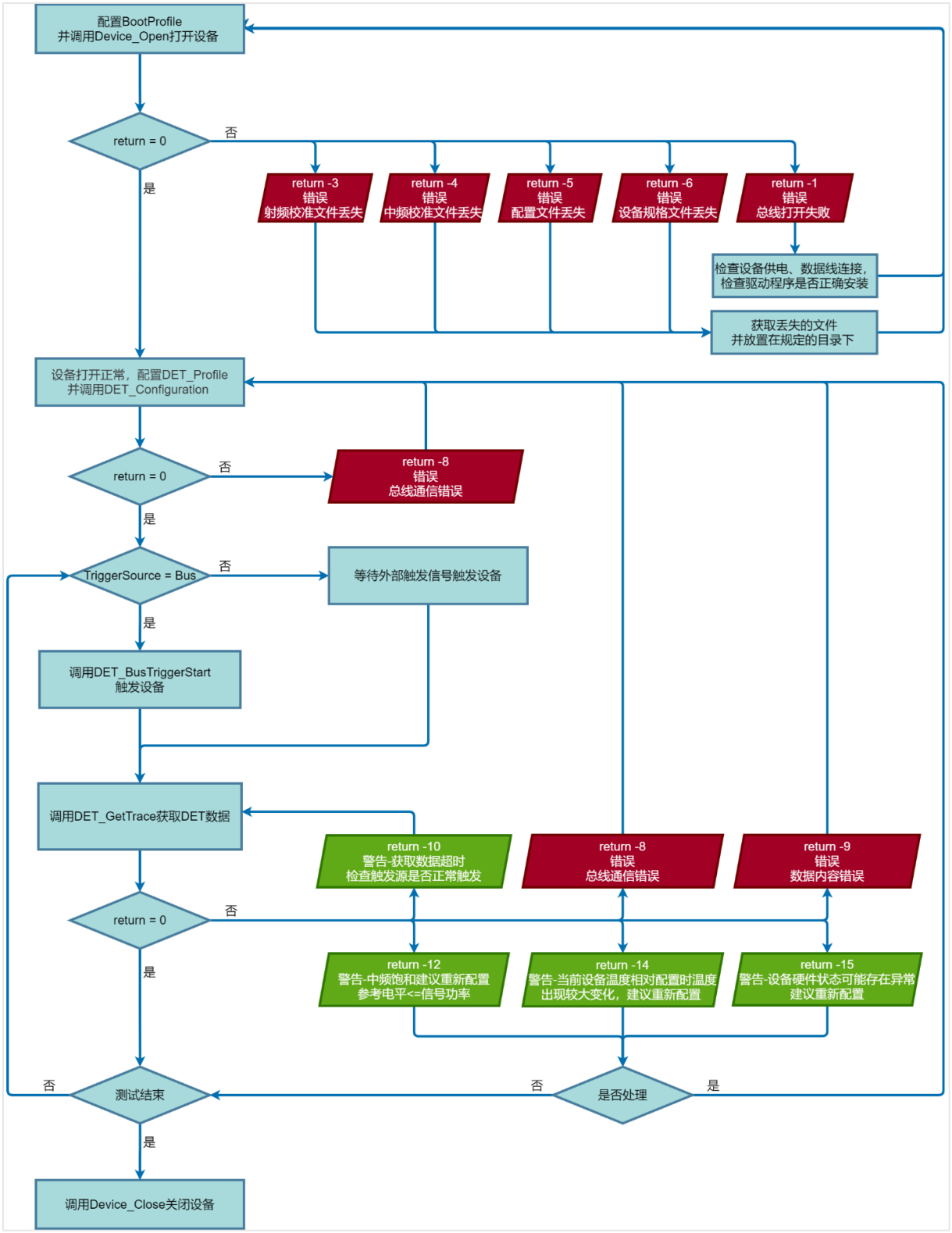

图5　DET 模式调用流程图(触发模式为 Fixed)

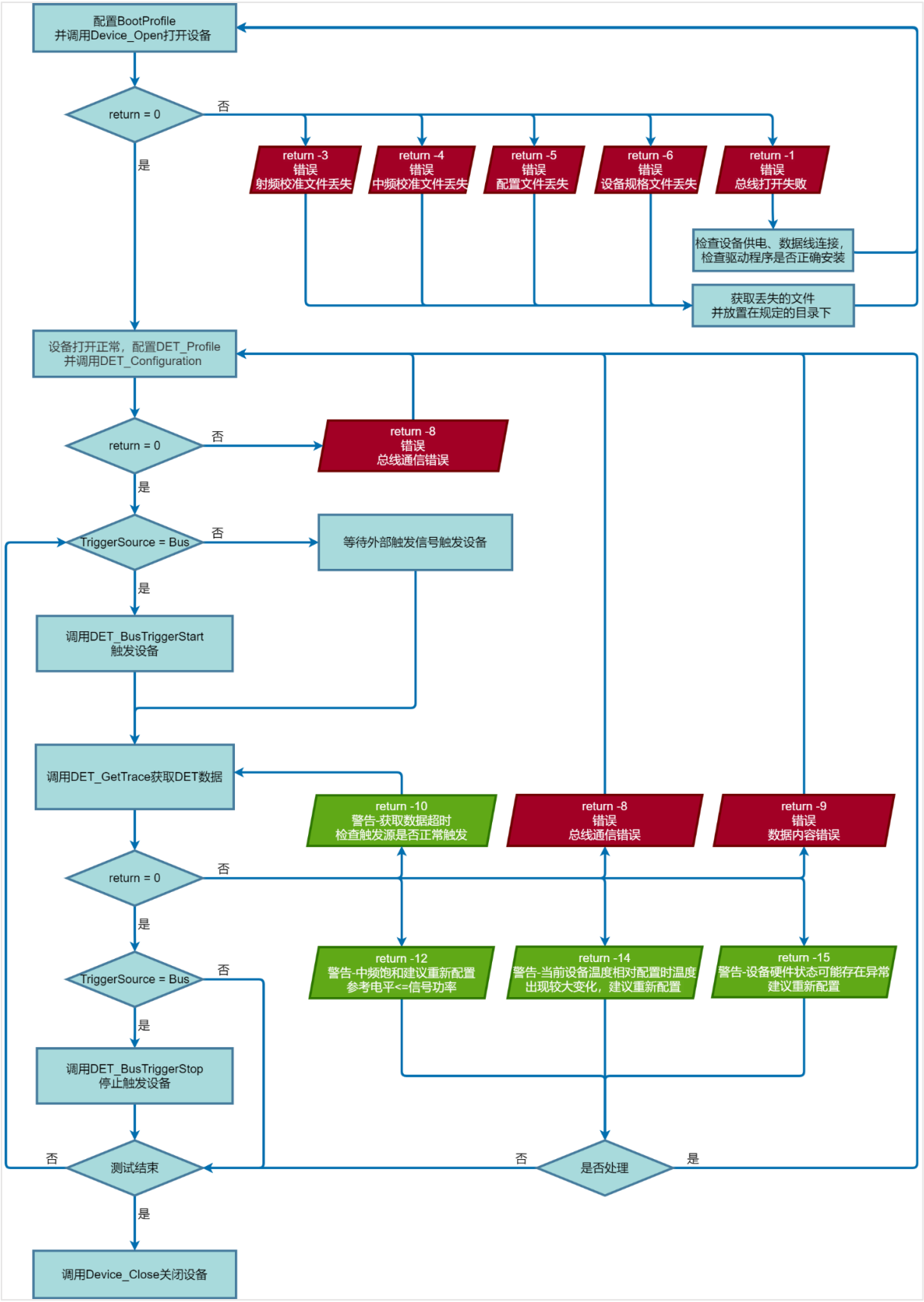

图6　DET 模式调用流程图(触发模式为 Adaptive)

5.4 实时频谱分析（RTA）的 API 调用地图

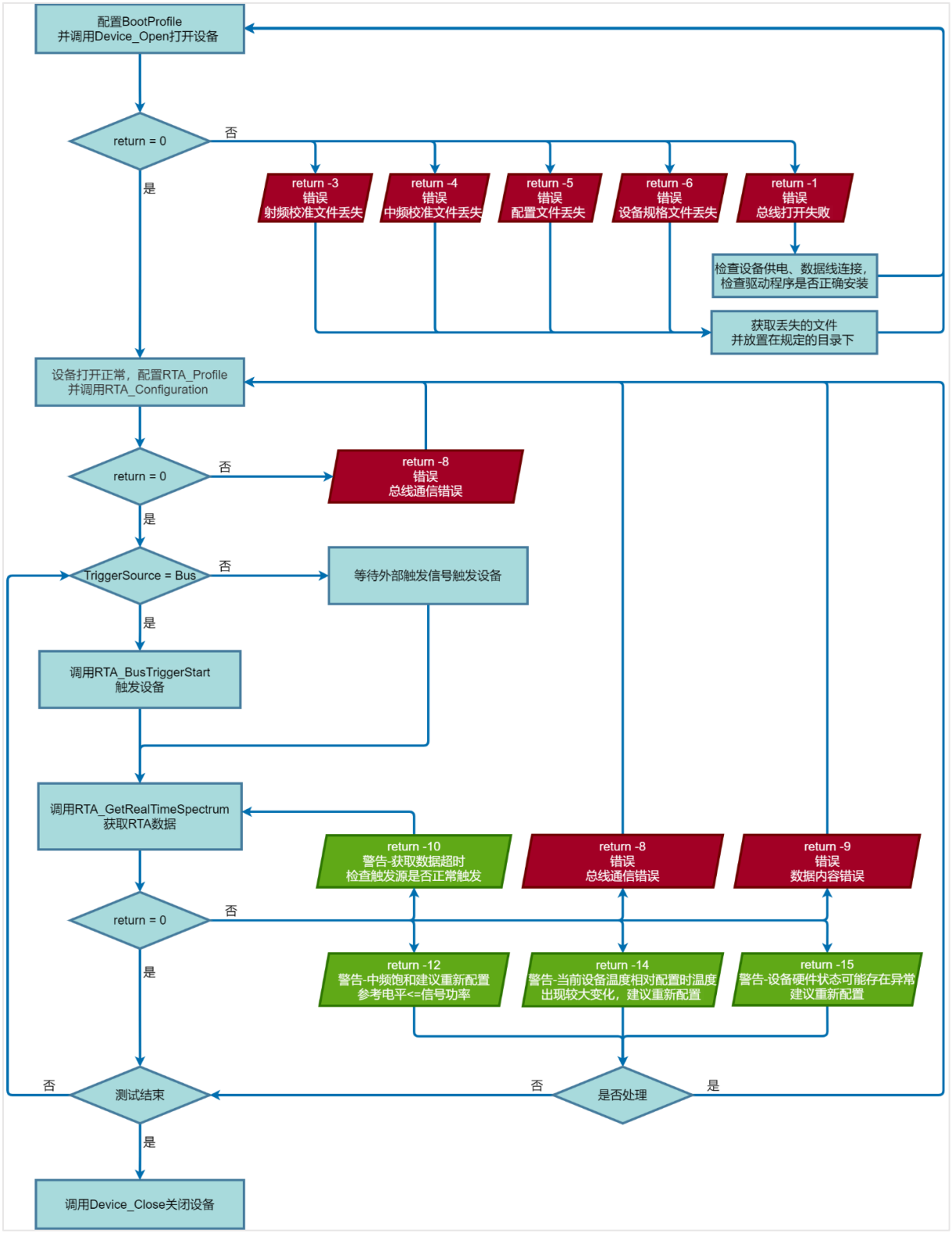

图7　RTA 模式调用流程图(触发模式为 Fixed)

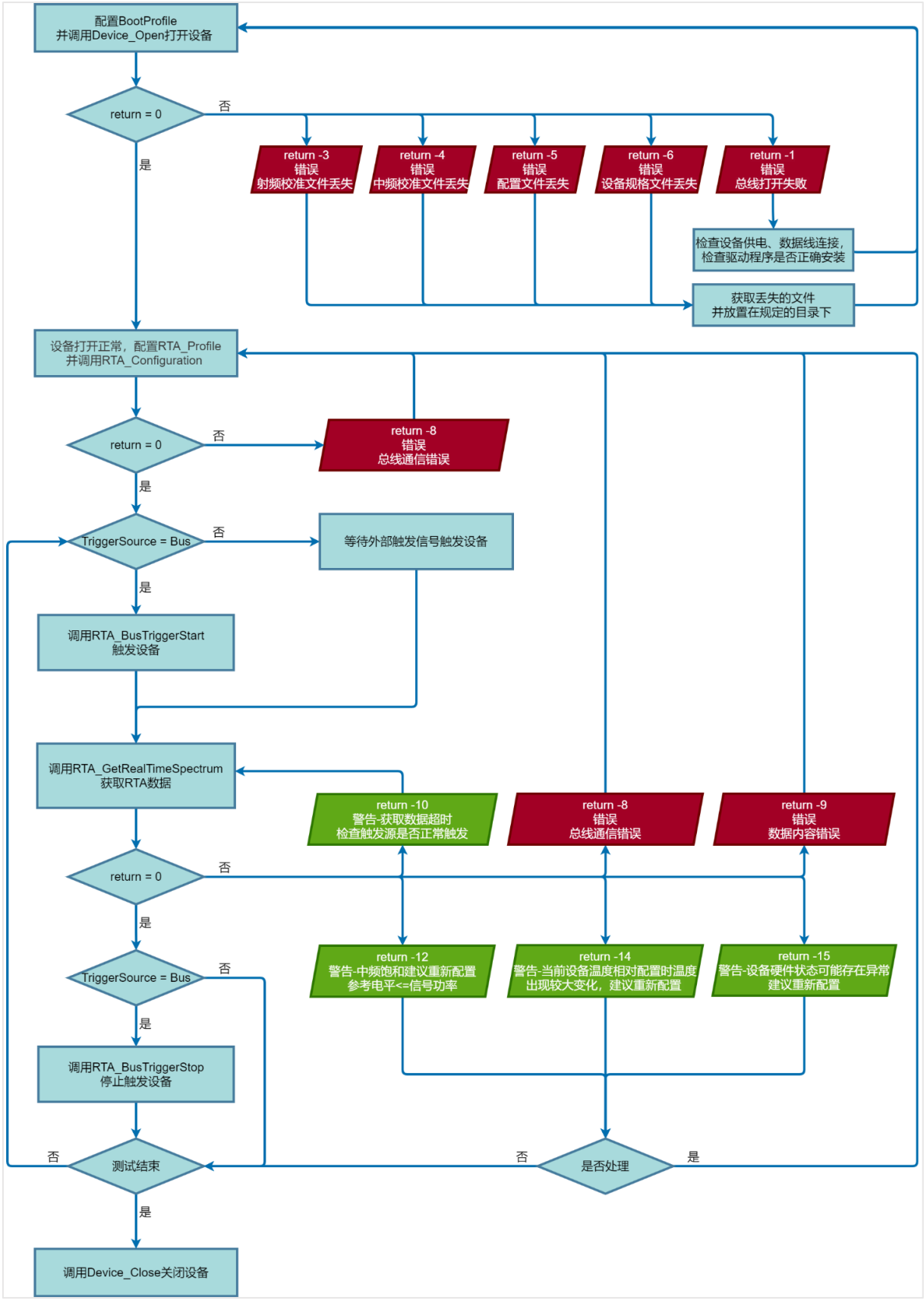

图8　RTA 模式调用流程图(触发模式为 Adaptive)

6 重要变量/重要设置及其概念

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

6.1 系统

表2 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3 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4　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	SWP	

	TracePointStrategy		在SWP模式下，系统的频谱分析方式由TracePointStrategy决定： 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，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： 对两端数据进行截取，以匹配所需的频率范围或设置 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。
4	迹线点数 TracePoints		
5	迹线对齐 TraceAlign		

6.4 分析

表5　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。该功能能有效抑制大部分组合分量杂散，但不会改善系统的剩余响应，同时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，可显著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 Flat Top 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≤5 kHz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（≤5 kHz）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。

6.4.1 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。

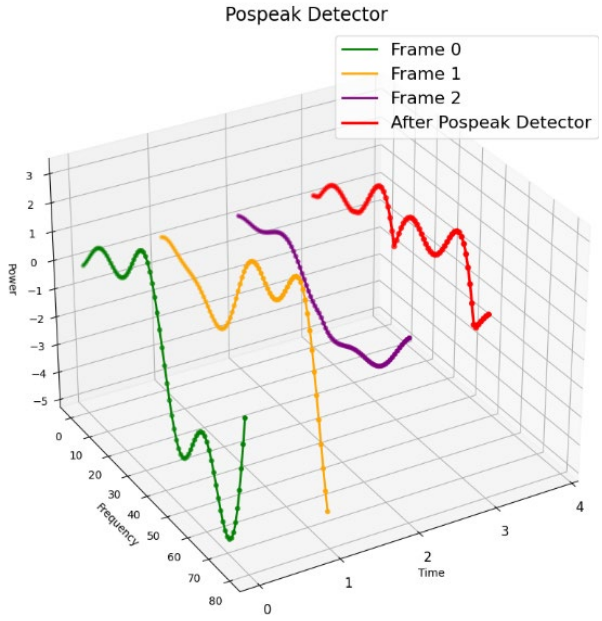

图9　正峰值检波

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。

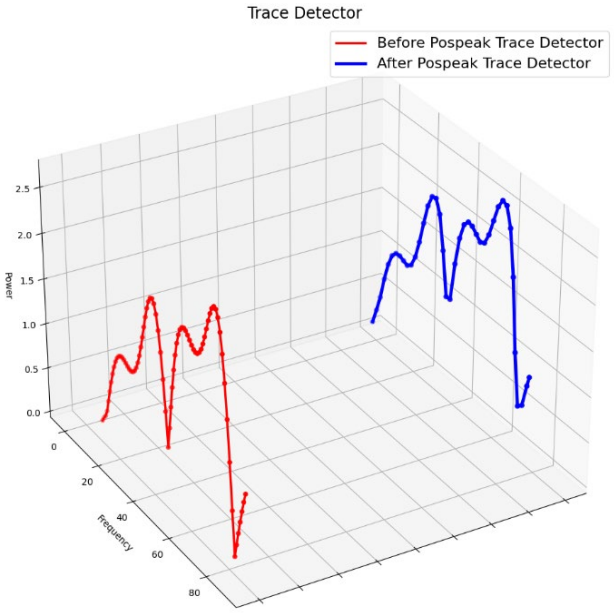

图10　正峰值迹线检波

6.5 默认单位

表6　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_Open

int Device_Open(void Device, int DeviceNum, const BootProfile_TypeDef* BootProfile,BootInfo_TypeDef* BootInfo)**	
功能描述	
未打开的设备需要被打开才能进行后续API的调用。调用本函数以打开设备，函数将返回一个句柄，该句柄用于在后续API调用中指向目标设备。当有多台设备时，通过指定不同的设备号（DeviceNum）来分别打开对应的设备，并获取其设备句柄。在后续API调用中，使用设备句柄来指定哪台设备被操作。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。在调用其它API时，通过此句柄来索引需要调用的设备。
int DeviceNum	指定设备号，当主机连接有多台设备时，可通过此设备号来选择相应的设备，设备号从0开始累加。
BootProfile_TypeDef* BootProfile	设置启动配置。
BootInfo_TypeDef* BootInfo	启动信息反馈。
BootProfile_TypeDef 详细定义	
PhysicalInterface_TypeDef PhysicalInterface	指定以何种物理端口来尝试设备的开启，必须正确设置才能正常开启设备，否则将无法打开设备。一台设备可能配备多种接口。 USB：使用USB接口进行数据传输。 ETH：使用100M/1000M 以太网进行数据传输；
DevicePowerSupply_TypeDef DevicePowerSupply	指定设备的供电方式，必须正确设置才能正常开启设备，否则将无法打开设备。 USBPortAndPowerPort：使用USB数据端口及独立电源端口双供电； USBPortOnly：仅使用USB数据端口供电； Others：当使用非USB总线时，比如ETH，使用此选项。
IPVersion_TypeDef ETH_IPVersion	以太网IP版本： IPv4：使用IPv4地址；

uint8_t ETH_IPAddress[16]	当物理接口为ETH时，用于指定IP地址，例如目标IP地址为192.168.1.100，ETH_IPAddress[0] = 192；ETH_IPAddress[1] = 168；ETH_IPAddress[2] = 1；ETH_IPAddress[3] = 100； 当网络协议为IPv4时，只需要填前4个数即可，数组其余部分不需要填。
uint16_t ETH_RemotePort	当物理接口为ETH时，指定侦听端口。
int32_t ETH_ErrorCode	当物理接口为ETH时，返回连接过程中的错误代码。
int32_t ETH_ReadTimeOut	当物理接口为ETH时，设定通信的超时时间。当设备配置与数据函数在此时间内未响应时，函数将强制返回并返回错误代码。
BootInfo_TypeDef 详细定义	
DeviceInfo_TypeDef DeviceInfo	设备信息
uint32_t BusSpeed	总线带宽信息
uint32_t BusVersion	总线固件版本。
uint32_t APIVersion	当前API版本。采用主版本，子版本，修订版本表示。 bit[31..16]表示主版本，bit[15..8]表示子版本，bit[7..0]表示修订。
int ErrorCodes[7]	启动过程中的错误代码清单。
int Errors	启动过程中的错误总数。
int WarningCodes[7]	启动过程中的警告代码清单。
int Warnings	启动过程中的警告总数。
DeviceInfo_TypeDef 详细定义	
uint64_t DeviceUID	设备独有的ID号。
uint16_t Model	设备型号。
uint16_t HardwareVersion	设备硬件版本。
uint16_t MFWVersion	设备主控固件版本。
uint16_t FFWVersion	设备FPGA固件版本。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	必须在其它函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的Device_Open调用，必须在整个模块使用需求结束之后，调用Device_Close函数，以释放内存。
示例	请参考Device_QueryDeviceInfo_Realtime() 函数的相关示例。

7.2 Device_Close

int Device_Close(void** Device)	
功能描述	
关闭已打开频谱仪设备，在结束调用设备时需要调用此函数已关闭USB设备及释放设备所开辟的内存空间。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后USB设备连接关闭，内存空间释放。如需要重新使用设备，则需要再次通过调用Device_Open，来建立USB连接及打开设备。
示例	请参考Device_QueryDeviceInfo_Realtime() 函数的相关示例。

7.3 Device_QueryDeviceState

int Device_QueryDeviceState(void** Device, DeviceState_TypeDef* DeviceState)	
功能描述	
获取当前频谱仪的设备状态信息，包括设备温度、硬件工作状态、地理时间信息（需要选件支持）等，非实时方式，不打断数据获取，但信息只在获取数据包后更新。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
DeviceState_TypeDef* DeviceState	当前设备状态相关信息的结构体指针。调用此函数后，该指针指向的相关信息会被更新成最新的值。
DeviceState_TypeDef 详细定义	
int16_t Temperature	设备的温度，摄氏度 = 0.01 * Temperature
double AbsoluteTimeStamp	绝对时间戳。
float Latitude	纬度坐标,北纬为正数，南纬为负数，以此区分南北纬。
float Longitude	经度坐标,东经为正数，西经为负数，以此区分东西经。
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_QueryDeviceInfo_Realtime()函数的相关示例。

7.4 Device_QueryDeviceState_Realtime

int Device_QueryDeviceState_Realtime(void** Device, DeviceState_TypeDef* DeviceState)	
功能描述	
获取设备状态，包括设备温度、硬件工作状态、地理时间信息（需要选件支持）等，实时获取，短时间内会占用数据通道。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
DeviceState_TypeDef* DeviceState	请参考 Device_QueryDeviceState ()函数同名结构体参数详细定义。
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_QueryDeviceInfo_Realtime()函数的相关示例。

7.5 Device_QueryDeviceInfo

int Device_QueryDeviceInfo(void** Device, DeviceInfo_TypeDef* DeviceInfo)	
功能描述	
获取设备信息，包括设备序列号及软硬件版本等相关信息，非实时方式，不打断数据获取，但信息只在获取数据包后更新。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
DeviceInfo_TypeDef* DeviceInfo	返回当前设备序列号及软硬件版本等相关信息的结构体指针。调用此函数后，该指针指向的相关信息会被更新成最新的值。 请参考 Device_Open ()函数同名结构体参数详细定义。
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_QueryDeviceInfo_Realtime()函数的相关示例。

7.7 Device_QueryDeviceInfo_Realtime

int Device_QueryDeviceInfo_Realtime(void** Device, DeviceInfo_TypeDef* DeviceInfo)	
功能描述	
获取设备信息，包括设备序列号及软硬件版本号等相关信息，实时获取，短时间内会占用数据通道，但可以保证总是获得即时的设备信息。	
兼容性	0.55.0及之后版本支持
参数说明	同函数Device_QueryDeviceInfo
void** Device	设备句柄。
DeviceInfo_TypeDef* DeviceInfo	返回当前设备序列号及软硬件版本等相关信息的结构体指针。调用此函数后，该指针指向的相关信息会被更新成最新的值。 请参考 Device_Open ()函数同名结构体参数详细定义。
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_Realtime(&Device, &DeviceState); DeviceInfo_TypeDef DeviceInfo; Status = Device_QueryDeviceInfo_Realtime(&Device, &DeviceInfo); Status = Device_Close(&Device);	

8 设备与系统 Device 其他函数

8.1 Device_SetSysPowerState

int Device_SetSysPowerState(void** Device, SysPowerMode_TypeDef SysPowerMode)	
功能描述	
设置设备的电源状态，包括上电运行、射频部分下电、射频部分待机等。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
SysPowerMode_TypeDef SysPowerMode	设备功耗控制： PowerOn: 系统所有工作区均上电； RFPowerOFF: 射频处于下电状态，不可快速唤醒。
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 = PowerON;
Status = Device_SetSysPowerState(&Device, SysPowerMode);
Status = Device_Close(&Device);
```

8.2 Device_SetFanState

int Device_SetFanState(void** Device, const FanState_TypeDef FanState, const float ThreshouldTemperature)	
功能描述	
控制设备风扇工作模式。（仅N45、N60、M60、M80设备支持）	
兼容性	0.55.0及之后版本支持

参数说明	
void** Device	设备句柄。
const FanState_TypeDef FanState	风扇模式： FAN_FORCE_ON:强制常开； FAN_FORCE_OFF:强制常关； FAN_AUTO:自动模式。
const float 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);
```

8.3 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	
通过外触发信号或系统内GNSS1PSS校准参考时钟频率。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
ClkCalibrationSource_TypeDef ClkCalibrationSource	选择校准所使用的定时信号源： CalibrateByExternal: 通过外触发进行Clock校准；

	CalibrateByGNSS1PPS = 0x01: 通过GNSS-1PPS进行Clock校准。
const double TriggerPeriod_s	输入已知的校准信号的准确周期，该值精度将直接影响校准效果。
const uint64_t TriggerCount	指定用于校准的触发次数，对于使用GNSS1PPS进行校准的场景，触发次越数多，通常对于抖动误差的抑制就越好，校准效果越好，但校准时间也越长。使用GNSS1PSS时，建议触发次数大于30，即校准超过30秒。
const bool RewriteRFCal	0：校准结果不写入校准文件，设备下电后校准结果失效； 1：校准结果写入校准文件，设备上下电后仍然保持校准状态。
double* RefCLKFreq_Hz	反馈本次校准得到的新的参考时钟频率，即校准结果。
返回值	0：无异常；非0：异常，详见附录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_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ETH接口的设备时，获取网络中所有设备的IP地址和子网掩码等信息。	
兼容性	0.55.0及之后版本支持
参数说明	
uint8_t*DeviceCount	设备数量。
NetworkDeviceInfo_TypeDef DeviceInfo[64]	返回设备序列号、设备类型和设备版本等设备信息。

uint8_t LocalIP[4]	返回本地IP地址。
uint8_t LocalMask[4]	返回本地子网掩码。
NetworkDeviceInfo_TypeDef 结构体定义	
uint64_t DeviceUID	设备序列号。
uint16_t Model	设备类型。
uint16_t HardwareVersion	硬件版本。
uint32_t MFWVersion	MCU固件版本。
uint32_t FFWVersion	FPGA固件版本。
uint8_t IPAddress[4]	IP地址。
uint8_t SubnetMask[4]	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);	

8.5 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ETH接口的设备时，通过设备UID，配置网络设备的IP地址和子网掩码。

兼容性	0.55.0及之后版本支持
参数说明	
const uint64_t DeviceUID	设备序列号。
const uint8_t IPAddress[4]	输入要配置的IP地址。
const uint8_t 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);
```

8.6 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	
通过NX系列设备现有的IP地址，为设备设置新的IP地址和子网掩码。	
兼容性	0.55.0及之后版本支持
参数说明	
const uint8_t DeviceIP[4]	输入现在的IP地址。
const uint8_t IPAddress[4]	输入要修改的IP地址。
const uint8_t SubnetMask[4]	输入要修改的子网掩码。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	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);	

8.7 Device_GetFullUID

int Device_GetFullUID (void** Device, uint64_t* UID_L64, uint32_t* UID_H32)	
功能描述	
获取完整UID信息。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
uint64_t* UID_L64	设备对应的UID的低64位。
uint32_t* UID_H32	设备对应的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);
```

8.8 Device_GetHardwareState

int Device_GetHardwareState (void** Device, HardWareState_TypeDef* HardWareState)	
功能描述	
获取硬件状态信息。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
HardWareState_TypeDef* HardWareState	返回GNSS 外设类型、设备是否支持信号源功能、设备是否支持ADC可变采样率等信息。
HardWareState_TypeDef 结构体定义	
GNSSPeriphType_TypeDef GNSSPeriphType	GNSS外设类型。 GNSS_None = 0：无外设； GNSS_For_EIO = 1：EIO； GNSS_For_NX = 2：NX； GNSS_For_PX = 3：PX。
GNSSType_TypeDef GNSSType	GNSS接收机类型。 None_GPS = 0：无GPS接收机； GNSS_GPS = 1：标准GPS； GNSS_GPS_Pro = 2：高性能GPS。
OCXOType_TypeDef OCXOType	GNSS上的OCXO类型。 None_OCXO = 0：无OCXO； GNSS_OCXO = 1：GNSS上的普通OCXO； GNSS_DOCXO = 2：GNSS上的可驯服OCXO。

uint8_t InternalOCXO	设备内部参考时钟是否是恒温晶振。
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）。
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);	

8.9 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。	
兼容性	0.55.0及之后版本支持
参数说明	
int DeviceNum	指定设备号，当主机连接有多台设备时，可通过此设备号来选择相应的设备，设备号从0开始累加。
const BootProfile_TypeDef* BootProfile	设置启动配置。 请参考Device_Open()函数同名结构体参数详细定义。
BootInfo_TypeDef* BootInfo	返回启动配置。 请参考Device_Open()函数同名结构体参数详细定义。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	调用此函数要保证设备处于未打开状态，即需要在Device_Open前调用此函数。

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);

8.10 Device_SetFreqScan

int Device_SetFreqScan (void** Device, double StartFreq_Hz, double StopFreq_Hz, uint16_t SweepPts)	
功能描述	
配置中心频率扫描参数。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
double StartFreq_Hz	起始频率，单位Hz。
double StopFreq_Hz	终止频率，单位Hz。
uint16_t SweepPts	需要扫描的频点数。
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);	

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	
在使用GNSS功能时，调用此函数可设置GNSS天线状态。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const GNSSAntennaState_TypeDef GNSSAntennaState	设置GNSS天线状态： 请参考Device_GetGNSSAntennaState ()函数同名枚举参数详细定义。
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。

9.2 Device_GetGNSSAntennaState

int Device_GetGNSSAntennaState(void** Device, GNSSAntennaState_TypeDef* GNSSAntennaState)	
功能描述	
在使用GNSS功能时，调用此函数可以以非实时的方式获取GNSS天线状态（需要选件支持），即不打断数据获取，但信息只在获取数据包后更新。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSSAntennaState_TypeDef GNSSAntennaState	设置GNSS天线状态： GNSS_AntennaExternal：外部天线； GNSS_AntennaInternal：内部天线。
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); GNSSAntennaState_TypeDef GNSSAntennaState = GNSS_AntennaExternal; Status = Device_SetGNSSAntennaState(&Device, GNSSAntennaState); Status = Device_GetGNSSAntennaState(&Device, &GNSSAntennaState); Status = Device_Close(&Device);

9.3 Device_GetGNSSAntennaState_Realtime

int Device_GetGNSSAntennaState_Realtime(void** Device, GNSSAntennaState_TypeDef* GNSSAntennaState)	
功能描述	
在使用GNSS功能时，调用此函数可以以实时的方式获取GNSS天线状态（需要选件支持），但实时方式短时间内会占用数据通道。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSSAntennaState_TypeDef GNSSAntennaState	设置GNSS天线状态： 请参考Device_GetGNSSAntennaState ()函数同名枚举参数详细定义。
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); GNSSAntennaState_TypeDef GNSSAntennaState = GNSS_AntennaExternal; Status = Device_SetGNSSAntennaState (&Device, GNSSAntennaState); Status = Device_GetGNSSAntennaState_Realtime (&Device, &GNSSAntennaState);	

Status = Device_Close(&Device);

9.4 Device_GetGNSSAltitude

int Device_GetGNSSAltitude(void** Device, int16_t* Altitude)	
功能描述	
在使用GNSS功能时，调用此函数可以以非实时的方式获取GNSS天线所处位置的海拔信息。（需要选件支持），即不打断数据获取，但信息只在获取数据包后更新。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
int16_t* Altitude	返回GNSS所处位置的海拔。
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; int16_t Altitude = 0; BootProfile_TypeDef BootProfile; BootProfile.DevicePowerSupply = USBPortAndPowerPort; BootProfile.PhysicalInterface =USB; BootInfo_TypeDef BootInfo; Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo); Status = Device_GetGNSSAltitude (&Device, &Altitude); Status = Device_Close(&Device);	

9.5 Device_AnysisGNSSTime

int Device_AnysisGNSSTime(double ABSTimestamp, int16_t* hour, int16_t* minute, int16_t* second, int16_t* Year, int16_t* month, int16_t* day)	
功能描述	
调用此函数可以解析GNSS时间日期信息。（需要选件支持）	
兼容性	0.55.0及之后版本支持
参数说明	
double ABSTimestamp	返回当前数据包对应的绝对时间戳。
int16_t* hour	返回GNSS时间日期信息中的时。

int16_t* minute	返回GNSS时间日期信息中的分。
int16_t* second	返回GNSS时间日期信息中的秒。
int16_t* Year	返回GNSS时间日期信息中的年。
int16_t* month	返回GNSS时间日期信息中的月。
int16_t* day	返回GNSS时间日期信息中的日。
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 ABSTimestamp = 0;int16_t hour = 0, minute = 0, second = 0, Year = 0, month = 0, day = 0;
Status = Device_AnysisGNSSTime (ABSTimestamp,&hour,&minute, &second, &Year, &month, &day);
Status = Device_Close(&Device);	

9.6 Device_SetDOCXOWorkMode

int Device_SetDOCXOWorkMode(void** Device, const DOCXOWorkMode_TypeDef DOCXOWorkMode)	
功能描述	
在使用GNSS功能时，调用此函数可以设置DOCXO工作状态。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const DOCXOWorkMode_TypeDef DOCXOWorkMode	设置DOCXO天线状态： DOCXO_LockMode：驯服模式； DOCXO_HoldMode：跟踪模式。
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_GetDOCXOWorkMode_Realtime函数相关示例。

9.7 Device_GetDOCXOWorkMode

int Device_GetDOCXOWorkMode(void** Device, const DOCXOWorkMode_TypeDef* DOCXOWorkMode)	
功能描述	
在使用GNSS功能时，调用此函数可以以非实时方式获取GNSS中DOCXO工作状态（需要选件支持），即不打断数据获取，但信息只在获取数据包后更新。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const DOCXOWorkMode_TypeDef DOCXOWorkMode	设置DOCXO天线状态： 请参考Device_SetDOCXOWorkMode ()函数同名枚举参数详细定义。
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_GetDOCXOWorkMode(&Device, &DOCXOWorkMode); Status = Device_Close(&Device);	

9.8 Device_GetDOCXOWorkMode_Realtime

int Device_GetDOCXOWorkMode_Realtime(void** Device,DOCXOWorkMode_TypeDef* DOCXOWorkMode)
功能描述
在使用GNSS功能时，调用此函数可以以实时的方式获取GNSS中DOCXO工作状态（需要选件支持），但实时方式短时间内会占用数据通道。

兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
DOCXOWorkMode_TypeDef* DOCXOWorkMode	设置DOCXO天线状态： 请参考Device_SetDOCXOWorkMode ()函数同名枚举参数详细定义。
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_GetDOCXOWorkMode_Realtime(&Device, &DOCXOWorkMode); Status = Device_Close(&Device);	

9.9 Device_GetGNSSInfo

int Device_GetGNSSInfo(void** Device, GNSSInfo_TypeDef* GNSSInfo)	
功能描述	
在使用GNSS功能时，调用此函数可以以非实时的方式获取GNSS设备状态（需要选件支持）。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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSSInfo_TypeDef* GNSSInfo	GNSS获取设备信息。
GNSSInfo_TypeDef 结构体定义	
float latitude	返回GNSS天线的经度。
Float longitude	返回GNSS天线的纬度。
int16_t altitude	返回GNSS天线的海拔。
uint8_t SatsNum	返回GNSS天线当前使用卫星数量。

uint8_t GNSS_LockState	返回GPS锁定状态。
uint8_t DOCXO_LockState	返回GDOCXO锁定状态。
DOCXOWorkMode_TypeDef DOCXO_WorkMode	返回DOCXO工作状态。 请参考Device_SetDOCXOWorkMode ()函数同名参数详细定义。
GNSSAntennaState_TypeDef GNSSAntennaState	返回天线状态。 请参考Device_GetGNSSAntennaState ()函数同名结构体参数详细定义。
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); GNSSInfo_TypeDef GNSSInfo; Status = Device_ GetGNSSInfo (&Device, & GNSSInfo); Status = Device_Close(&Device);	

9.10 Device_GetGNSSInfo_Realtime

int Device_GetGNSSInfo_Realtime(void** Device, GNSSInfo_TypeDef* GNSSInfo)	
功能描述	
在使用GNSS功能时，调用此函数可以以实时的方式获取GNSS设备状态（需要选件支持）。实时方式实时获取，但短时间内会占用数据通道。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSSInfo_TypeDef* GNSSInfo	GNSS获取设备信息。 请参考Device_GetGNSSInfo ()函数同名结构体参数详细定义。
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);
GNSSInfo_TypeDef GNSSInfo;
Status = Device_GetGNSSInfo_Realtime(&Device, &GNSSInfo);
Status = Device_Close(&Device);
```

9.11 Device_GetGNSS_SatDate

int Device_GetGNSS_SatDate (void** Device, GNSS_SatDate_TypeDef* GNSS_SatDate)	
功能描述	
获取GNSS信噪比（需要选件支持），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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSS_SatDate_TypeDef* GNSS_SatDate	返回GNSS 信噪比信息。
GNSS_SatDate_TypeDef 结构体定义	
uint8_t SatsNum_All	当前范围可视卫星。
uint8_t SatsNum_Use	用于定位的卫星数量。
GNSS_SNR_TypeDef GNSS_SNR_UsePos	用于定位的卫星信噪比信息。 Max_SatxC_No：最大信噪比； Min_SatxC_No：最小信噪比； Avg_SatxC_No：平均信噪比。
GNSS_SNR_TypeDef GNSS_SNR_NotUsePos	在视野内，但未用于定位的卫星信噪比信息。
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);
GNSS_SatDate_TypeDef GNSS_SatDate;
Status = Device_GetGNSS_SatDate(&Device, &GNSS_SatDate);
Status = Device_Close(&Device);
```

9.12 Device_GetGNSS_SatDate_Realtime

int Device_GetGNSS_SatDate_Realtime (void** Device, GNSS_SatDate_TypeDef* GNSS_SatDate)	
功能描述	
获取GNSS信噪比（需要选件支持），实时获取，短时间内会占用数据通道。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
GNSS_SatDate_TypeDef* GNSS_SatDate	返回GNSS 信噪比信息。 请参考Device_GetGNSS_SatDate函数同名结构体参数详细定义。 注意：仅在GNSS锁定后，获取的信噪比和卫星数量才是有效值，默认值是0。
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);
GNSS_SatDate_TypeDef 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	0.55.0 及之后版本支持
参数说明	
void** Device	设备句柄。
SWP_Profile_TypeDef* UserProfile_O	输出默认的配置参数集合。
SWP_Profile_TypeDef 详细定义	
double StartFreq_Hz	起始频率，单位Hz。
double StopFreq_Hz	终止频率，单位Hz。
double CenterFreq_Hz	中心频率，单位Hz。
double Span_Hz	频率扫宽，单位Hz。
double RefLevel_dBm	参考电平，单位dBm。
double RBW_Hz	分辨率带宽，单位Hz。
double VBW_Hz	视频带宽，单位Hz。
double SweepTime	当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。
SWP_FreqAssignment_TypeDef FreqAssignment	设置频率的指定方式，选择以StartStop或CenterSpan设定频率。 StartStop：以起始频率、终止频率指定扫描范围； CenterSpan：以中心频率、扫宽指定扫描范围。
Window_TypeDef Window	指定FFT分析所使用的窗函数： FlatTop：具有良好的幅度准确度； Blackman_Nuttall：具有良好的频率分辨力； LowSideLobe：具有良好的抑制频谱泄漏能力。
RBWMode_TypeDef RBWMode	设置RBW更新方式： 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。
VBWMode_TypeDef VBWMode	设置VBW更新方式: VBW_Manual: 手动输入VBW； VBW_EqualToRBW: 强制 VBW = RBW； VBW_TenPercentRBW: 强制 VBW = 0.1 * RBW； VBW_OnePercentRBW: 强制 VBW = 0.01 * RBW； VBW_TenTimesRBW: 强制 VBW = 10 * RBW，完全旁路VBW滤波器。
SweepTimeMode_TypeDef SweepTimeMode	扫描时间模式： 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。
Detector_TypeDef Detector	设置检波器： 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。
TraceFormat_TypeDef TraceFormat	设置迹线格式： TraceFormat_Standard：频率等间隔； TraceFormat_PrecisFrq：频率准确。
TraceDetectMode_TypeDef TraceDetectMode	设置迹线检波模式（频率轴）： TraceDetectMode_Auto：自动选择迹线检波模式； TraceDetectMode_Manua：指定迹线检波模式。
TraceDetector_TypeDef TraceDetector	设置迹线检波器类型： 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。
uint32_t TracePoints	设置期望的迹线点数，系统会根据设置的扫描范围和RBW进行自动调整，并返回实际可用且最接近的迹线点数。
TracePointsStrategy_TypeDef TracePointsStrategy	设置迹线点数的设置策略： SweepSpeedPreferred：优先保证扫描速度最快，尽量靠近设置的目标迹线点数； PointsAccuracyPreferred：优先保证实际迹线点数接近设置的目标迹线点数； BinSizeAssigned：优先保证按照设定的迹线点数来生成迹线。
TraceAlign_TypeDef TraceAlign	设置迹线对齐的方式： NativeAlign：自然对齐； AlignToStart：对齐至起始频率。
FFTExecutionStrategy_TypeDef FFTExecutionStrategy	设置FFT执行策略： Auto：根据设置自动选择使用CPU还是FPGA进行FFT计算； Auto_CPUPreferred：根据设置自动选择使用CPU还是FPGA进行FFT计算，CPU优先； Auto_FPGAPreferred：根据设置自动选择使用CPU还是FPGA进行FFT计算，FPGA优先；

	CPUOnly_LowResOcc：强制使用CPU计算，低资源占用，最大FFT点数256K； CPUOnly_MediumResOcc：强制使用CPU计算，中资源占用，最大FFT点数1M； CPUOnly_HighResOcc：强制使用CPU计算，高资源占用，最大FFT点数4M； FPGAOnly：强制使用FPGA计算。
RxPort_TypeDef RxPort	设置信号接收端口：（仅M60、M80、N60、N45支持） 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。
SpurRejection_TypeDef SpurRejection	设置杂散抑制： Bypass：不进行杂散抑制； Standard：中级杂散抑制； Enhanced：高级杂散抑制； 杂散抑制等级越高，扫描速率越慢。
ReferenceClockSource_TypeDef ReferenceClockSource	设置参考时钟源： ReferenceClockSource_Internal：内部参考时钟（默认10MHz）； ReferenceClockSource_External：外部参考时钟（默认10MHz），当外部参考无法锁定时将自动切换为内部参考； ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：强制使用外部参考时钟，即使无法锁定也不会切换至内部参考。
double ReferenceClockFrequency	设置参考时钟频率 Hz，可用手动对系统时钟准确度进行修正。
uint8_t EnableReferenceClockOut	设置使能参考时钟输出。（仅E90、E200、N400支持）
SWP_TriggerSource_TypeDef TriggerSource	设置接收机的扫频输入触发源： 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。
TriggerEdge_TypeDef TriggerEdge	设置输入触发边沿： RisingEdge：由上升沿触发； FallingEdge：由下降沿触发； DoubleEdge：由双边沿触发。

TriggerOutMode_TypeDef TriggerOutMode	设置触发输出的模式： None：无触发输出； PerHop：随每帧完成时输出； PerSweep：随每次扫描完成时输出； PerProfile：随每次配置切换输出。
TriggerOutPulsePolarity_TypeDef TriggerOutPulsePolarity	设置触发输出的脉冲极性： Positive：正脉冲； Negative：负脉冲。
uint32_t PowerBalance	设置SWP模式下的动态功耗控制。典型范围为0~5000，增大该值可降低功耗但会降低扫描速度。
GainStrategy_TypeDef GainStrategy	设置增益策略： LowNoisePreferred：侧重低噪声； HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef Preamplifier	设置前置放大器动作： AutoOn：自动使能前置放大器； ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。
int8_t Atten	设置衰减dB,设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
SWP_TraceType_TypeDef TraceType	设置输出迹线类型： 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。
LOOptimization_TypeDef LOOptimization	设置本振优化： 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。
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的频谱扫描模式（SWPMode）并配置SWP模式的相关参数。SWP模式下的频率、参考电平、分辨率带宽等参数统一封装在SWP_Profile_TypeDef结构体中。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const SWP_Profile_TypeDef* SWP_ProfileIn	配置集输入。 请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
SWP_Profile_TypeDef* SWP_ProfileOut	配置集输出。 请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
SWP_TraceInfo_TypeDef* TraceInfo	返回频谱迹线的相关信息。
SWP_TraceInfo_TypeDef 详细定义	
int FullsweepTracePoints	完整迹线的点数。
int PartialsweepTracePoints	每个频点的迹线点数，即每次GetPart的点数。
int TotalHops	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最为近的频率点
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	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	视频检波的检波比。
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（单位为秒）
double EstimateMinSweepTime	当前配置下，所能设定的最小扫描时间（单位为秒，结果主要受Span、RBW、VBW、帧扫描时间等因素影响）。
DataFormat_TypeDef DataFormat	时域数据格式。
uint64_t SamplePoints	时域数据采样长度。
uint32_t GainParameter	增益相关参数，包括Space(31~24Bit)、PreAmplifierState(23~16Bit)、StartRFBand(15~8Bit)、StopRFBand(7~0Bit)。
DSPPlatform_Typedef DSPPlatform	当前配置所使用的DSP运算平台。 CPU_DSP：在CPU进行计算； FPGA_DSP：在FPGA进行计算。
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	
将频谱仪设备的频谱扫描模式（SWPMode）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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
SWPApplication_TypeDef Application	在SWP模式，根据应用目标给出推荐设备配置： 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。
const SWP_Profile_TypeDef* SWP_ProfileIn	配置集输入。

SWP_Profile_TypeDef* SWP_ProfileOut	配置集输出。
SWP_TraceInfo_TypeDef* TraceInfo	返回频谱迹线的相关信息。
uint8_t ifDoConfig	返回下发成功与否结果
SWP_Profile_TypeDef 详细定义	请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
SWP_TraceInfo_TypeDef 详细定义	请参考 SWP_Configuration ()函数同名结构体参数详细定义。
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
可获取SWP模式下在每个跳频点上获取的频谱结果，同时返回跳频频点序号、帧序号和测量数据的辅助信息，以便用户自己将多次的扫描的结果拼接成整个频谱曲线，并返回函数状态。

兼容性	0.55.0及之后版本支持

参数说明	
void** Device	设备句柄。
double Freq_Hz[]	返回的频率数组。数组大小等于TraceInfo.PartialSweepTracePoints。
float PowerSpec_dBm[]	返回的幅度数组。数组大小等于TraceInfo.PartialSweepTracePoints。
int* HopIndex	返回数据的跳频频点序号。
int* FrameIndex	返回数据的帧序号。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回数据的辅助信息。
MeasAuxInfo_TypeDef 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，设备内定时器提供。
double AbsoluteTimeStamp	绝对时间戳。系统内GNSS提供。
float Latitude	纬度坐标,北纬为正数，南纬为负数，以此区分南北纬。
float Longitude	经度坐标,东经为正数，西经为负数，以此区分东西经。
返回值	0：无异常；非0：异常，详见附录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, DevNum, &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_GetFullSweep(void** Device, double Freq_Hz[], float PowerSpec_dBm[], MeasAuxInfo_TypeDef* MeasAuxInfo)	
功能描述	
可获取SWP模式下一整条频谱结果和获取测量数据的辅助信息，并同时返回函数状态。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
double Freq_Hz[]	返回的频率数组。数组大小等于TraceInfo.FullSweepTracePoints。
float PowerSpec_dBm[]	返回的幅度数组。数组大小等于TraceInfo.FullSweepTracePoints。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考 SWP_GetPartialSweep ()函数同名结构体参数详细定义。
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);
```

```
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
SWPTrace_TypeDef* PartialTrace	返回包含配置、返回信息和暂存数据的顶层结构体。
SWPTrace_TypeDef 详细定义	
double* Freq_Hz	暂存在Device指针中的Frequency数据的地址。
float* PowerSpec_dBm	暂存在Device指针中的PowerSpec_dBm数据的地址。
int HopIndex	数据的跳频频点索引，用于拼接频谱。
int FrameIndex	数据的帧索引，用于准正峰值检波等应用。
void* AlternIQStream	时域数据（交织IQ形式）的地址。
float ScaletoV	时域数据至电压绝对值（V）的系数。
MeasAuxInfo_TypeDef MeasAuxInfo	返回测量数据的辅助信息。 请参考 SWP_GetPartialSweep ()函数同名结构体参数详细定义。
SWP_Profile_TypeDef SWP_Profile	数据的配置信息。 请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
SWP_TraceInfo_TypeDef SWP_TraceInfo	数据的迹线信息。 请参考 SWP_Configuration ()函数同名结构体参数详细定义。
DeviceInfo_TypeDef DeviceInfo	数据的设备信息。 请参考 Device_Open ()函数同名结构体参数详细定义。
DeviceState_TypeDef DeviceState	数据对应的设备状态。 请参考 Device_QueryDeviceState ()函数同名结构体参数详细定义。
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_MaxHoldReset(void Device)**	
功能描述	
当迹线模式TraceType为 MaxHold或MinHold时，重置保持的迹线数据。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	无。
调用约束	仅在TraceType为 MaxHold或MinHold时生效。
示例	

```
int Status = -1;int DeviceNum = 0;void* Device = NULL;
BootProfile_TypeDef BootProfile;
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(), &HopIndex,&FrameIndex,
&MeasAuxInfo);
SWP_ResetTraceHold(&Device);
Status = Device_Close(&Device);
```

12 相位噪声测量模式 Phase Noise

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。	
兼容性	0.55.58及之后版本支持
参数说明	
void** Device	设备句柄。
PNM_Profile_TypeDef* PNM_Profile	相位噪声测量的配置结构体。
PNM_Profile_TypeDef 详细定义	
double CenterFreq	中心频率。
float Threshold	载波判决门限。
double RBWRatio	RBW比例（各段RBW/各段起始频率）。
double StartOffsetFreq	起始频偏。
double StopOffsetFreq	终止频偏。
uint32_t TraceAverage	迹线平滑次数。
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。	
兼容性	0.55.58及之后版本支持
参数说明	
void** Device	设备句柄。
const PNM_Profile_TypeDef* PNM_Profile_in	相位噪声测量的配置结构体（输入）。 请参考PNM_ProfileDeInit()函数同名结构体参数详细定义。

PNM_Profile_TypeDef* PNM_Profile_out	相位噪声测量的配置结构体（返回）。 请参考PNM_ProfileDeInit()函数同名结构体参数详细定义。
PNM_MeasInfo_TypeDef* PNM_MeasInfo	相位噪声测量的测量信息结构体。
PNM_MeasInfo_TypeDef 详细定义	
uint32_t Segments	频率分段数（十倍频程划分）。
uint32_t TracePoints	迹线点数。
uint32_t PartialUpdateCounts	一次分析对应的迹线刷新次数（Get接口调用次数）。
uint32_t FramesInSegment[PHASENOISE_ MAXFREQSEGMENT]	各分段的总帧数。
uint32_t FrameDetRatioOfSegment[PHAS ENOISE_MAXFREQSEGMENT]	各分段的帧检波比。
double StartFreqOfSegment[PHASENOIS E_MAXFREQSEGMENT]	各分段的起始频率。
double StopFreqOfSegment[PHASENOIS E_MAXFREQSEGMENT]	各分段的终止频率。
double RBWOfSegment[PHASENOISE_M AXFREQSEGMENT]	各分段的RBW。
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。	
兼容性	0.55.58及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录1。

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。

12.4 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	
void** 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	
void** Device	设备句柄。
double* CarrierFreq	载波频率。
float* CarrierPower	载波功率。
double Freq[]	迹线频率轴（以Hz为单位）。
float PhaseNoise[]	迹线功率轴（以dBc/Hz为单位）。
uint32_t 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）。
MeasAuxInfo_TypeDef* MeasAuxInfo	辅助测量信息结构体。 请参考SWP_GetPartialSweep()函数同名结构体参数详细定义。
float* 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_Configuration之后进行调用。
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	
高级功能：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	
void** Device	设备句柄。
double* CarrierFreq	载波频率。
uint8_t* Found	载波存在标志。
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	
高级功能：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	
void** Device	设备句柄。
PNM_MeasInfo_TypeDef* MeasInfo	复位帧检波比后，更新相位噪声测量的测量信息结构体。 请参考PNM_Configuration()函数同名结构体参数详细定义。
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	
高级功能：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	
void** Device	设备句柄。
uint32_t 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）。
PNM_MeasInfo_TypeDef* MeasInfo	调整帧检波比后，更新相位噪声测量的测量信息结构体。 请参考PNM_Configuration()函数同名结构体参数详细定义。
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);
// 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。
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);
// 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 IQ 数据记录 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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
IQS_Profile_TypeDef *UserProfile_O	IQS配置结构体指针，为输入/输出变量。
IQS_Profile_TypeDef 详细定义	
double CenterFreq_Hz	设置中心频率。
double RefLevel_dBm	设置参考电平。
uint32_t DecimateFactor	设置时域数据的抽取倍数，有效分析带宽 = 原始分析带宽/抽取倍数。
RxPort_TypeDef RxPort	设置信号接收端口： ExternalPort：接收机接收来自外部输入端口输入的数据； InternalPort：接收机接收来自内部的辅助信号源的射频信号。 （仅M60、M80、N60、N45支持）
uint32_t BusTimeout_ms	设置数据传输超时时间（ms），如果设备在调用数据获取函数后于此时间内未能成功获取数据则函数返回错误。
IQS_TriggerSource_TypeDef TriggerSource	设置输入触发源： 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：在GNSS-1PPS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GNSS模块选件）。
TriggerEdge_TypeDef TriggerEdge	设置输入触发边沿： RisingEdge：上升沿触发； FallingEdge：下降沿触发； DoubleEdge：双边沿触发。
TriggerMode_TypeDef TriggerMode	设置输入触发模式： FixedPoints：触发后，采集定点长度（TriggerLength）的数据； Adaptive：触发后，持续采集数据，直到收到终止信号（外触发终止边沿或总线触发终止指令）。
uint64_t TriggerLength	设置触发长度，即每次触发所采集的数据点数，当TriggerMode = FixedPoints时生效。
double TriggerLevel_dBm	设置电平触发门限。当TriggerSource = Level时生效。
double TriggerLevel_SafeTime	设置电平触发防抖安全时间，单位为s。若触发信号无法在该时间内保持有效电平触发将不被执行。当TriggerSource = Level时生效。
double TriggerDelay	设置触发延迟，单位为s。触发后，触发动作将在此延时之后执行。
double PreTriggerTime	设置预触发时间，单位为s。触发后，触发动作前该预触发时长内的数据也将被包含在触发数据中。
TriggerTimerSync_TypeDef TriggerTimerSync	设置触发定时器的同步： NoneSync：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GNSS模块选件） SyncToGNSS1PPS_FallingEdge：定时器触发与GNSS-1PPS下降沿同步；（需GNSS模块选件）

	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GNSS模块选件）； 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GNSS模块选件）。
double TriggerTimer_Period	设置定时触发周期，单位秒。仅在 TriggerSource = Timer 时生效。
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。
double ReTrigger_Period	自动重触发，设置再触发的触发周期，单位s。
uint16_t ReTrigger_Count	自动重触发，设置每次触发后，再触发的执行次数。
DataFormat_TypeDef DataFormat	设置IQ数据的输出数据格式： Complex8bit：IQ两路数据为8位格式； Complex16bit：IQ两路数据为16位格式； Complex32bit：IQ两路数据为32位格式。
GainStrategy_TypeDef GainStrategy	设置增益策略： LowNoisePreferred：侧重低噪声； HighLinearityPreferred：侧重高线性度。
PreamplifierState_TypeDef Preamplifier	设置前置放大器动作： AutoOn：自动使能前置放大器； ForcedOff：强制保持前置放大器关闭。
uint8_t AnalogIFBWGrade	设置模拟中频带宽档位。
uint8_t IFGainGrade	设置中频增益档位。档位越高中频增益越高。
ReferenceClockSource_TypeDef ReferenceClockSource	设置参考时钟源： ReferenceClockSource_Internal：内部参考时钟（默认10MHz）； ReferenceClockSource_External：外部参考时钟（默认10MHz），当外部参考无法锁定时将自动切换为内部参考； ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：强制使用外部参考时钟，即使无法锁定也不会切换至内部参考。
double ReferenceClockFrequency	设置参考时钟频率 Hz。
uint8_t EnableReferenceClockOut	设置使能参考时钟输出。（仅E90、E200、N400支持）

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。
int8_t Atten	设置衰减dB，设定频谱仪通道衰减量，默认-1（自动）。当该值不等于-1（自动）时，其优先于RefLevel_dBm。
DCCancelerMode_TypeDef DCCancelerMode	设置直流抑制器模式： DCCOff：直流抑制关闭； DCCHighPassFilterMode：高通滤波器模式（更好的直流抑制效果，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； 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。
QDCMode_TypeDef QDCMode	设置IQ幅相修正器模式： QDCOff: 关闭QDC功能； QDCManualMode：开启并使用手动模式；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； 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。
float QDCIGain	设置归一化线性增益 I路，1.0 表示无增益，设置范围 0.8~1.2。QDCMode = QDCManualMode时生效。
float QDCQGain	设置归一化线性增益 Q路，1.0 表示无增益，设置范围 0.8~1.2。QDCMode = QDCManualMode时生效。
float QDCPhaseComp	设置相位补偿系数，设置范围 -0.2~+0.2。QDCMode = QDCManualMode 时生效。
int8_t DCCIOffset	设置I通道直流偏置，LSB。DCCancelerMode = DCCManualOffsetMode 时生效。
int8_t DCCQOffset	设置Q通道直流偏置，LSB。DCCancelerMode = DCCManualOffsetMode 时生效。
LOOptimization_TypeDef	设置本振优化：

LOOptimization	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。
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	
将频谱仪设备配置成IQ时域数据流模式(IQS)。IQS模式下，本振信号固定，并接收以本振频率为中心一定带宽的射频信号。同时在抽取倍数>2时，使用USB3.0及高速硬盘进行数据传输可实现连续时域记录功能。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const IQS_Profile_TypeDef* IQS_ProfileIn	IQS配置结构体指针，为输入变量。 请参考 IQS_ProfileDeInit ()函数同名结构体参数详细定义。
IQS_Profile_TypeDef* IQS_ProfileOut	IQS配置结构体指针，为输出变量。 请参考 IQS_ProfileDeInit ()函数同名结构体参数详细定义。
IQS_StreamInfo_TypeDef* StreamInfo	IQS时域数据流模式下，IQ数据流的相关信息。
IQS_StreamInfo_TypeDef 详细定义	
double Bandwidth	当前配置对应的接收机物理通道或数字信号处理的带宽。
double IQSampleRate	当前配置对应的IQ单路采样率，单位S/s（Sample/second）。
uint64_t PacketCount	当前配置对应的总数据包数，仅在FixedPoints模式下生效
uint64_t StreamSamples	Fixedpoints模式下表示当前配置对应采样的总点数；Adaptive模式下没有物理意义，值为0。
uint64_t StreamDataSize	Fixedpoints模式下表示当前配置对应采样的总字节数；Adaptive模式下没有物理意义，值为0。

uint32_t PacketSamples	每次调用IQS_GetIQStream 获取到的数据包中采样点数 每个数据包中包含的样点数。
uint32_t PacketDataSize	每次调用IQS_GetIQStream 所得到的有效数据字节数。
uint32_t GainParameter	增益相关参数，包括Space(31~24Bit)、PreAmplifierState(23~16Bit)、StartRFBand(15~8Bit)、StopRFBand(7~0Bit)。
返回值	0：无异常；非0：异常，详见附录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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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	
将终止当前总线触发。当配置TriggerMode = FixedPoints时，总线触发在由IQS_BusTriggerStart函数发起并达到指定触发长度后会自行终止，而无需调用该函数。	
兼容性	0.55.0 及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非 0：异常，详见附录 1。
调用约束	需要在 IQS_GetIQStream 之后进行调用。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
void** AlternIQStream	时域数据（交织IQ形式）的地址。一个数据包为固定64968个字节。当选择的IQ数据类型为int8_t时，I、Q两路分别为32484个点，每个点占1个字节；当选择的IQ数据类型为int16_t时，I、Q两路分别为16242个点，每个点占2个字节；当选择的IQ数据类型为int32_t时，I、Q两路分别为8121个点，每个点占4个字节。IQ数据是按照IQIQ…的排列方式存储。
float* ScaleToV	时域数据至电压绝对值（V）的系数。
IQS_TriggerInfo_TypeDef* TriggerInfo	IQ数据流的触发相关信息。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考 IQS_ProfileDeInit ()函数同名结构体参数详细定义。

IQS_TriggerInfo_TypeDef 详细定义	
uint64_t SysTimerCountOfFirstDataPoint	首个数据点对应的系统时间计数器值。
uint16_t InPacketTriggeredDataSize	数据中有效触发数据的字节数。
uint16_t InPacketTriggerEdges	数据中所包含的边沿个数。
uint32_t StartDataIndexOfTriggerEdges[25]	各触发边沿的数据起始位置。
uint64_t SysTimerCountOfEdges[25]	各触发边沿的系统时间戳。
int8_t EdgeType[25]	各触发边沿的极性。
返回值	0：无异常；非0：异常，详见附录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;
IQS_Profile_TypeDef 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 IQ 数据记录 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。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const IQS_Profile_TypeDef* ProfileIn	IQS配置结构体指针，为输入变量。 请参考 IQS_ProfileDeInit ()函数同名结构体参数详细定义。
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	
调用此函数使能多机同步运行。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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和int32_t，根据用户需要自行选择数据格式。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
IQStream_TypeDef* IQStream	IQ数据流，包括IQ数据及相关配置信息等。
IQStream_TypeDef 详细定义	
void* AlternIQStream	返回时域IQ数据包。一整个数据包为固定64968个字节。当选择的IQ数据类型为int8_t时，I、Q两路分别为32484个点，每个点占1个字节；当选择的IQ数据类型为int16_t时，I、Q两路分别为16242个点，每个点占2个字节；当选择的IQ数据类型为int32_t时，I、Q两路分别为8121个点，每个点占4个字节。IQ数据是按照IQIQ…的排列方式存储。
float IQS_ScaleToV	时域数据至电压绝对值（V）的系数。
float MaxPower_dBm	数据中的功率最大值。
uint32_t MaxIndex	功率最大值在数据中的索引。
IQS_Profile_TypeDef IQS_Profile	数据的配置信息。 请参考 IQS_ProfileDeInit ()函数同名结构体参数详细定义。
IQS_StreamInfo_TypeDef StreamInfo	数据的格式信息。 请参考 IQS_Configuration ()函数同名结构体参数详细定义。

IQS_TriggerInfo_TypeDef TriggerInfo	数据的触发信息。 请参考 IQS_GetIQStream ()函数同名结构体参数详细定义。
DeviceInfo_TypeDef DeviceInfo	数据的设备信息。 请参考 Device_Open ()函数同名结构体参数详细定义
DeviceState_TypeDef DeviceState	数据的设备状态信息。 请参考 Device_QueryDeviceState ()函数同名结构体参数详细定义。
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，其中时域数据格式为int8_t、int16_t和int32_t，根据用户需要自行选择数据格式。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
IQStream_TypeDef* IQStream	IQ数据流，包括IQ数据及相关配置信息等。

	请参考IQS_GetIQStream_PM1函数同名结构体参数详细定义。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考SWP_GetPartialSweep()函数同名结构体参数详细定义。
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	
调用此函数获取数据类型为int16_t的IQ时域数据。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	此参数设备句柄。
int16_t IQ_data[]	用于接收单路数据16位的IQ数据数组。
返回值	0：无异常；非0：异常，详见附录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); //创建 I 路数据数组。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
DET_Profile_TypeDef *UserProfile_O	DET配置结构体指针，为输入/输出变量。
DET_Profile_TypeDef 详细定义	请参考IQS_ProfileDeInit()函数同名结构体参数详细定义。
double CenterFreq_Hz	
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： 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_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。
uint16_t DET_TraceDetectRatio	设置DET迹线检波比。
GainStrategy_TypeDef GainStrategy	请参考IQS_ProfileDeInit()函数同名结构体参数详细定义。
PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
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	
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const DET_Profile_TypeDef* DET_ProfileIn	DET配置结构体指针，为输入变量。 请参考 DET_ProfileDeInit ()函数同名结构体参数详细定义。
DET_Profile_TypeDef* DET_ProfileOut	DET配置结构体指针，为输出变量。 请参考 DET_ProfileDeInit ()函数同名结构体参数详细定义。
DET_StreamInfo_TypeDef* StreamInfo	DET模式下，DET数据的相关信息。
DET_StreamInfo_TypeDef 详细定义	
uint64_t PacketCount	单次触发的数据包总数。
uint64_t StreamSamples	单次触发的总数据点数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。
uint64_t StreamDataSize	单次触发的总数据字节数，TriggerMode = Fixedpoints 时有效，TriggerMode = Adaptive时无效，值为0。

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	增益参数信息。 Bit31~24：表示增益空间； Bit23~16：表示前置放大器状态； Bit15~0：表示频段信息。
返回值	0：无异常；非0：异常，详见附录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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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	0.55.0及之后版本支持
参数说明	

void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在DET_GetPowerStream之后进行调用。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
float NormalizedPowerStream[]	$\sqrt{I^2+Q^2}$的值。
float* ScaleToV	NormalizedPowerStream至电压绝对值（V）的系数。
DET_TriggerInfo_TypeDef* TriggerInfo	DET数据的触发相关信息。 请参考 IQS_GetIQStream ()函数同名结构体参数详细定义。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考 SWP_GetPartialSweep ()函数同名结构体参数详细定义。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在DET_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;
DET_TriggerInfo_TypeDef  TriggerInfo;
MeasAuxInfo_TypeDef  MeasAuxInfo;
Status = DET_BusTriggerStart(&Device);
Status = DET_GetPowerStream(&Device, NormalizedPowerStream.data(), &ScaleToV, &TriggerInfo, &MeasAuxI
nfo);
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在DET_Configuration之后进行调用，且TriggerSource选择Timer。
示例	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
ZSP_Profile_TypeDef* UserProfile_O	ZeroSpan配置结构体指针，为输入/输出变量。
ZSP_Profile_TypeDef 详细定义	
double CenterFreq_Hz	请参考DET_ProfileDeInit()函数同名结构体参数详细定义。
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 DET_TraceDetectRatio	
GainStrategy_TypeDef GainStrategy	
PreamplifierState_TypeDef Preamplifier	
uint8_t AnalogIFBWGrade	
uint8_t IFGainGrade	
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

double VBW_Hz	视频带宽。
VBWMode_TypeDef VBWMode	VBW更新方式。
RBWFilterType_TypeDef RBWFilterType	RBW滤波器类型。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const ZSP_Profile_TypeDef* ProfileIn	ZSP配置结构体指针，为输入变量。 请参考 ZSP_ProfileDeInit ()函数同名结构体参数详细定义。
ZSP_Profile_TypeDef* ProfileOut	ZSP配置结构体指针，为输出变量。 请参考 ZSP_ProfileDeInit ()函数同名结构体参数详细定义。
DET_StreamInfo_TypeDef* StreamInfo	ZSP模式下，ZSP数据的相关信息。 请参考 DET_Configuration ()函数同名结构体参数详细定义。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
RTA_Profile_TypeDef *UserProfile_O	RTA配置结构体指针，为输入/输出变量。
RTA_Profile_TypeDef 详细定义	
double CenterFreq_Hz	请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
double RefLevel_dBm	
double RBW_Hz	
double VBW_Hz	
RBWMode_TypeDef RBWMode	
VBWMode_TypeDef VBWMode	
uint32_t DecimateFactor	设置抽取倍数。
Window_TypeDef Window	请参考 SWP_ProfileDeInit ()函数同名结构体参数详细定义。
SweepTimeMode_TypeDef SweepTimeMode	
double SweepTime	
Detector_TypeDef Detector	
TraceDetectMode_TypeDef TraceDetectMode	
TraceDetector_TypeDef TraceDetector	
uint32_t TraceDetectRatio	迹线检波检波比。迹线检波器将原始频谱迹线以每TraceDetectRatio个原始频谱数据点检出1个输出频谱数据点。
RxPort_TypeDef RxPort	请参考IQS_ProfileDeInit()函数同名结构体参数详细定义。
uint32_t BusTimeout_ms	

RTA_TriggerSource_TypeDef TriggerSource	
TriggerEdge_TypeDef TriggerEdge	
TriggerMode_TypeDef TriggerMode	
double TriggerAcqTime	设置输入触发后的采样时间，仅在FixedPoints模式下生效。
TriggerOutMode_TypeDef TriggerOutMode	请参考IQS_ProfileDeInit()函数同名结构体参数详细定义。
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	
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	
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
const RTA_Profile_TypeDef* RTA_ProfileIn	RTA配置结构体指针，为输入变量。 请参考RTA_ProfileDeInit()函数同名结构体参数详细定义。
RTA_Profile_TypeDef* RTA_ProfileOut	RTA配置结构体指针，为输出变量。 请参考RTA_ProfileDeInit()函数同名结构体参数详细定义。
RTA_FrameInfo_TypeDef* StreamInfo	RTA模式下，RTA数据的相关信息。
RTA_FrameInfo_TypeDef 详细定义	
double StartFrequency_Hz	频谱的起始频率。
double StopFrequency_Hz	频谱的终止频率。
double POI	100%截获概率下的信号最短持续时间,以s为单位。

double TraceTimestampStep	每包数据内各条Trace的时间戳步进。(包整体时间戳为TriggerInfo中的SysTimerCountOfFirstDataPoint)
double TimeResolution	每个时域数据的采样时间，也是时间戳的分辨率。
double PacketAcqTime	每包数据对应的采集时间。
uint32_t PacketCount	当前配置对应的总数据包数，仅在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FFT的点数。
uint32_t FrameWidth	FFT帧截取后的点数，也是数据包中每条Trace的点数，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。
uint32_t MaxDensityValue	概率密度位图的单个位点元素值上限。
uint32_t GainParameter	增益相关参数，包括Space(31~24Bit)、PreAmplifierState(23~16Bit)、StartRFBand(15~8Bit)、StopRFBand(7~0Bit)。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在RTA_ProfileDeInit之后进行调用。
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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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_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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
uint8_t SpectrumStream[]	返回由连续频谱帧组成的频谱流，表示为相对功率，LSB = 0.75dB。此数组大小等于通过RTA_Configuration函数得到的RTA_FrameInfo.PacketValidPoints。
RTA_PlotInfo_TypeDef* RTA_PlotInfo	RTA获取后返回的绘图信息结构体。
RTA_TriggerInfo_TypeDef* TriggerInfo	RTA数据的触发相关信息。 请参考 IQS_GetIQStream ()函数同名结构体参数详细定义。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考SWP_GetPartialSweep()函数同名结构体参数详细定义。
RTA_PlotInfo_TypeDef 详细定义	
float ScaleTodBm	

float OffsetTodBm	频谱帧幅度至dBm的比例因子与偏置值。频谱帧的幅度绝对功率等于SpectrumStream[] * ScaleTodBm + OffsetTodBm。
uint64_t SpectrumBitmapIndex	当前概率密度图片段在完整触发数据中的序号。
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);	

17.6 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频谱数据。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。

uint8_t SpectrumStream[]	返回由连续频谱帧组成的频谱流，表示为相对功率，LSB = 0.75dB。此数组大小等于通过RTA_Configuration函数得到的RTA_FrameInfo.PacketValidPoints。
uint16_t SpectrumBitmap[]	返回概率密度图位图。此数组大小等于通过RTA_Configuration函数得到的RTA_FrameInfo.FrameHeight * RTA_FrameInfo.FrameWidth。
RTA_PlotInfo_TypeDef* RTA_PlotInfo	RTA获取后返回的绘图信息结构体。 请参考 RTA_GetRealTimeSpectrum_Raw ()函数同名结构体参数详细定义。
RTA_TriggerInfo_TypeDef* TriggerInfo	RTA数据的触发相关信息。 请参考 IQS_GetIQStream ()函数同名结构体参数详细定义。
MeasAuxInfo_TypeDef* MeasAuxInfo	返回测量数据的辅助信息。 请参考SWP_GetPartialSweep()函数同名结构体参数详细定义。
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);
```

17.7 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	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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 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。

18.1 Demod_Check

int Demod_Check ()	
功能描述	
核验解调库是否存在。	
兼容性	0.55.55及之后版本支持
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; Status = Demod_Check (); 	

18.2 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	
void** Device	设备句柄。
返回值	0：存在；非0：不存在。
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。

18.3 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	
void** 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。

18.4 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	
void** 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。

18.5 Demod_GetVersion

int Demod_GetVersion (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	
void** Device	设备句柄。
char version[]	返回解调API版本。
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。

18.6 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	
Demod_Profile_TypeDef* DemodProfile	解调配置结构体。
Demod_Profile_TypeDef 详细定义	
uint64_t SamplePoints	采样点数。
double SampleRate	采样率，Hz。
double SymbolRate	符号率，sym/s。
Demod_ModType_TypeDef ModType	设置待解调信号的调制类型，支持FSK2/ FSK4/ GMSK/ BPSK/ QPSK/ PSK8/ QAM16/ ASK2/ QAM64/ AM/ FM/ PM/ CW/ LowerSideband/ UpperSideband/ QAM128/ QAM256。
Demod_FilterType_TypeDef FilterType	滤波器类型： RootRaisedCosine：根升余弦滤波器；
double FilterAlpha	滤波器滚降系数(目前仅支持根升余弦，0.01 <= Alpha <= 0.99)。
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。

18.7 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	
void** Device	设备句柄。
const Demod_Profile_TypeDef* DemodProfileIn	输入解调配置结构体。 请参考 Demod_DeInit ()函数同名结构体参数详细定义。

Demod_Profile_TypeDef* 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。 请参考 Demod_DeInit ()函数同名结构体参数详细定义。
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。

18.8 Demod_Execute

int Demod_Execute (void** Device,const IQStream_TypeDef* IQStream, DemodInfo_TypeDef* DemodInfo)	
功能描述	
执行解调功能。	
兼容性	0.55.55及之后版本支持
参数说明	
void** Device	设备句柄。
const IQStream_TypeDef* IQStream	IQ数据流，包括IQ数据及相关配置信息等。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
DemodInfo_TypeDef* 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。
DemodInfo_TypeDef* DemodInfo	输出解调信息结构体。
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、QAM。
double SNR	信噪比，单位dB，仅PSK、QAM。
double GainImb	IQ增益不平衡，单位dB，仅PSK、QAM。
double QuadError	IQ正交倾斜误差，单位角度，仅PSK、QAM。
double FSK_Deviation	FSK频偏，单位Hz。
double CarrPower	载波功率，单位dBm，仅ASK。
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。
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 = 100e3;
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.PacketSamples != 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);
```

18.9 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	
Demod_ModType_TypeDef ModType	调制类型。
Demod_SymbolMap_TypeDef 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。
uint32_t* MapNum	符号映射表中可用的符号个数。
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。
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);	

19 脉冲检测 Pulse Det（选件）

19.1 Pulse_Open

int Pulse_Open (void** Device)	
功能描述	
打开脉冲检测功能，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	
void** 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Pulse_Detect ()函数相关示例。

19.2 Pulse_Close

int Pulse_Close (void** Device)	
功能描述	
关闭脉冲检测功能。	
兼容性	0.55.55及之后版本支持
参数说明	
void** Device	设备句柄。
返回值	0：无异常；非0：异常，详见附录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。

19.3 Pulse_Detect

int Pulse_Detect(void** Device, const Pulse_Profile_TypeDef* Pulse_Profile, PulseInfo_TypeDef* PulseInfo)	
功能描述	
对获取的检波分析数据执行脉冲检测功能。	
兼容性	0.55.55及之后版本支持
参数说明	
void** Device	设备句柄。

const Pulse_Profile_TypeDef* Pulse_Profile	输入脉冲检测数据，包括：需要检测的数据、阈值等。
PulseInfo_TypeDef* PulseInfo	输出脉冲检测结果，包括：脉宽、周期、占空比等。
Pulse_Profile_TypeDef 详细定义	
uint32_t ExpPulseNum	期望获取的脉冲数量。
Unit_TypeDef unit	脉冲数据单位。 Voltage_V：V； Power_dBm：dBm。
float* Pulse	脉冲数据的起始内存地址，数据单位取决于unit。
uint64_t PulseSize	脉冲数据长度。
double TimeResolution_s	脉冲数据时间分辨率，单位秒s。
double DetThreshold	脉冲检测门限，单位与数据保持一致。
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。
PulseAMPParam_TypeDef* PulseAMPParam	脉冲检测幅度参数的起始内存地址。
PulseEstParam_TypeDef* PulseEstParam	脉冲检测绘图参数的起始内存地址。
PulseStatsParam_TypeDef PulseStats	脉冲检测统计参数。
PulseTDParam_TypeDef 详细定义	
double RiseTime	上升时间。
double RiseEdge	上升沿。
double FallTime	下降时间。
double FallEdge	下降沿。
double Width	脉宽。
double Period	周期。
float DutyCycle	占空比%。
PulseAMPParam_TypeDef 详细定义	
float TopLevel_dBm	峰值电平dBm。
float TopLevel_V	峰值电平V。

float BaseLevel_dBm	基准电平dBm。
float BaseLevel_V	基准电平V。
float TopToBaseRatio_dB	峰基比dB。
float TopToBaseDiff_V	峰基差V。
float Droop_dB	下垂dB。
float Droop_V	下垂V。
float Overshoot_dB	过冲dB。
float Overshoot_V	过冲V。
float Ripple_dB	波纹dB。
float Ripple_V	波纹V。
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	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dBm。
float* Noise_V	推测出的噪声数据起始内存地址V。
float* Signal_dBm	推测出的信号数据起始内存地址dBm。
float* Signal_V	推测出的信号数据起始内存地址V。
PulseStatsParam_TypeDe详细定义	
double MinPRI	最小周期。
double MaxPRI	最大周期。
double MeanPRI	平均周期。
double MinPW	最小脉宽。
double MaxPW	最大脉宽。
double MeanPW	平均脉宽。
float PRIDeviationPercent	周期偏差百分比。
float PWDeviationPercent	脉宽偏差百分比。
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 = 1;
DET_ProfileIn.TriggerMode = FixedPoints;
DET_ProfileIn.TriggerSource = Bus;
DET_ProfileIn.TriggerLength = 16242;
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, &S
   caleToV, &TriggerInfo, &MeasAuxInfo);
}
vector<float> NormalizedPowerStream_dBm(NormalizedPowerStream.size());//将 DET 数据的单位转换成 dBm
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);
```

19.4 Pulse_Detect_PM1

int Pulse_Detect_PM1 (void** Device, const Pulse_Profile_TypeDef* Pulse_Profile, PulseInfoPM1_TypeDef* PulseInfoPM1)	
功能描述	
对获取的IQ数据执行脉冲检测功能。	
兼容性	0.55.55及之后版本支持
参数说明	
void** Device	设备句柄。
const Pulse_Profile_TypeDef* Pulse_Profile	输入脉冲检测数据，包括：需要检测的数据、阈值等。 请参考Pulse_Detect()函数同名结构体参数详细定义。
PulseInfoPM1_TypeDef* 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。
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。 请参考Pulse_Detect()函数同名结构体参数详细定义。
PulseAMPParam_TypeDef* PulseAMPParam	脉冲检测幅度参数的起始内存地址。 请参考Pulse_Detect()函数同名结构体参数详细定义。
PulseEstParam_TypeDef* PulseEstParam	脉冲检测绘图参数的起始内存地址。 请参考Pulse_Detect()函数同名结构体参数详细定义。
PulseStatsParam_TypeDef PulseStats	脉冲检测统计参数。 请参考Pulse_Detect()函数同名结构体参数详细定义。
uint8_t* Mod	脉冲调制类型，0表示CW，1表示LFM。
PulseFreqPhaseParam_TypeDef* PulseFreqPhase	脉冲的频率和相位信息。

PulseFreqPhaseParam_TypeDef 详细定义	
double FreqMean	频率均值。
double FreqErrorRMS	频率误差。
double PhaseMean	相位均值。
double PhaseErrorRMS	相位误差。
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);
```

20 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，目前仅N45、N60、M60、M80支持添加此选件。

20.1 ASG_ProfileDeInit

int ASG_ProfileDeInit(void** Device, ASG_Profile_TypeDef* Profile)	
功能描述	
调用此函数初始化ASG功能的相关参数，初始化模拟信号源。	
兼容性	0.55.0及之后版本支持。
参数说明	
void** Device	设备句柄。
ASG_Profile_TypeDef* Profile	ASG配置结构体指针。
ASG_Profile_TypeDef 详细定义	
double CenterFreq_Hz	中心频率，单位为Hz，在信号源工作在SIG_Fixed模式下生效；输入范围1M-1GHz;步进1Hz。
double Level_dBm	输出功率，单位为dBm，在信号源工作在SIG_Fixed模式下生效；输入范围-127--5dBm；步进0.25dB。
double StartFreq_Hz	频率扫描模式下的起始频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StopFreq_Hz	频率扫描模式下的终止频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StepFreq_Hz	频率扫描模式下的步进频率，单位为Hz，在信号源工作在SIG_FreqSweep_*模式下生效；输入范围1M-1GHz;步进1Hz。
double StartLevel_dBm	功率扫描模式下的起始功率，单位为Hz。
double StopLevel_dBm	功率扫描模式下的终止功率，单位为Hz。
double StepLevel_dBm	功率扫描模式下的步进功率，单位为Hz。
double DwellTime_s	频率扫描模式 或 功率扫描模式 下，单位为s，当触发模式为BUS时，扫描驻留时间，单位为s，在信号源工作在*Sweep*模式下生效；输入范围0-1000000；步进1。
double ReferenceClockFrequency	指定参考频率：对内参考和外参考均生效。
ReferenceClockSource_TypeDef	选择参考时钟的输入源：

ReferenceClockSource	ReferenceClockSource_Internal：内部时钟源(默认10MHz)； ReferenceClockSource_External：外部时钟源(默认10MHz)，当系统检测到外部时钟无法锁定时，将自动切换至内部参考； ReferenceClockSource_Internal_Premium：内部时钟源-高品质，选择DOCXO或OCXO； ReferenceClockSource_External_Forced：外部时钟源，并且无视锁定情况，即使失锁也不会切换至内部参考。
ASG_Port_TypeDef Port	信号源输出端口： ASG_Port_External：外部端口； ASG_Port_Internal：内部端口。
ASG_Mode_TypeDef Mode	关闭、点频、频率扫描（外触发，同步至接收）、功率扫描（外触发，同步至接收）： ASG_Mute：静音； ASG_FixedPoint：定点； ASG_FrequencySweep：频率扫描； ASG_PowerSweep：功率扫描。
ASG_TriggerSource_TypeDef TriggerSource	信号源触发输入模式： ASG_TriggerIn_FreeRun：自由运行； ASG_TriggerIn_External：外触发； ASG_TriggerIn_Bus：定时器触发。
ASG_TriggerInMode_TypeDef TriggerInMode	信号源的触发模式： ASG_TriggerInMode_Null：自由运行； ASG_TriggerInMode_SinglePoint：单点触发（触发一次进行单次的频率或功率的配置）； ASG_TriggerInMode_SingleSweep：单次扫描触发（触发一次进行一个周期的扫描）； ASG_TriggerInMode_Continous：连续扫描触发（触发一次连续工作）。
ASG_TriggerOutMode_TypeDef TriggerOutMode	信号源的触发模式： ASG_TriggerOutMode_Null：自由运行； ASG_TriggerOutMode_SinglePoint：单点触发（一次跳频输出一个脉冲）； ASG_TriggerOutMode_SingleSweep：单次扫描触发（一次扫描输出一个脉冲）。
返回值	0：无异常；非0：异常，详见附录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。

20.2 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ASG功能的相关参数。ASG模式下的中心频率、功率、驻留时间等参数统一封装在ASG_Profile_TypeDef结构体中。	
兼容性	0.55.0及之后版本支持
参数说明	
void** Device	设备句柄。
ASG_Profile_TypeDef* ProfileIn	ASG配置结构体指针，为输入变量。 请参考ASG_ProfileDeInit ()函数同名结构体参数详细定义。
ASG_Profile_TypeDef* ProfileOut	ASG配置结构体指针，为输出变量。 请参考ASG_ProfileDeInit ()函数同名结构体参数详细定义。
ASG_Info_TypeDef* ASG_Info	ASG模式下，ASG相关信息，为输出变量。
ASG_Info_TypeDef 详细定义	
uint32_t SweepPoints	扫描点数。
返回值	0：无异常；非0：异常，详见附录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);
```

21 模拟信号解调 ASD

ADS为模拟调制解调处理接口，用户可调用其中的函数做模解调处理。

21.1 ASD_Open

void ASD_Open (void** AnalogMod)	
功能描述	
打开Analog功能，并在内存中分配一定空间以存储Analog的相关数据。在调用Analog的其它函数之前必须先调用此函数。当需要同时调用某一条函数时，可通过改变多加入几个Analog指针来进行操作。	
兼容性	0.55.0及之后版本支持
参数说明	
void** AnalogMod	运行Analog所需的内存空间引用。调用此函数后，函数将返回当前打开的Analog功能的内存地址。后续在调用其它API时，必须通过此引用来索引此次的地址。
返回值	无。
调用约束	在其它Analog函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的ASD_Open调用，必须在整个功能使用需求结束之后，调用ASD_Close函数，以释放内存。
示例	请参考 ASD_FMDemodulation ()函数相关示例。

21.2 ASD_Close

void ASD_Close (void** AnalogMod)	
功能描述	
关闭Analog功能，释放所开辟的内存空间。	
兼容性	0.55.0及之后版本支持
参数说明	
void** AnalogMod	运行Analog所需的内存空间引用。
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Analog功能关闭，内存空间释放。如需要重新调用Analog功能，则再次通过调用ASD_Open，打开Analog功能。
示例	请参考 ASD_FMDemodulation ()函数相关示例。

21.3 ASD_FMDemodulation

int ASD_FMDemodulation (void** AnalogMod , const IQStream_TypeDef* IQStreamIn, bool reset, float result[], FM_DemodParam_TypeDef* FM_DemodParam)	
功能描述	
对IQ数据进行FM解调。	
兼容性	0.55.0及之后版本支持
参数说明	
void** AnalogMod	运行Analog所需的内存空间引用。
const IQStream_TypeDef* IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
bool reset	重置缓存，对于一次连接解调来说，只有第一次调用函数的时候需要置成1，后面都是置0。
float result[]	FM解调后的信号数据，数据长度与IQ信号数据长度相同。
FM_DemodParam_TypeDef* FM_DemodParam	FM解调后返回FM调制信号频率、FM调制频偏等参数。
double carrierOffsetHz	载波的频率偏移。
FM_DemodParam_TypeDef 详细定义	
double ModRate	FM调制信号频率。
double ModDeviation	FM调制频偏。
double CarrierFreqOffset	载波偏移。
double ModRate_Avg	多次平均后的FM调制信号频率。
double ModDeviation_Avg	多次平均后的FM调制频偏。
double CarrierFreqOffset_Avg	多次平均后的载波偏移。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在ASD_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 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);
void* Analog = NULL; bool reset = 1; double carrierOffsetHz = 0;
ASD_Open(&Analog);
float* result = new float[IQStream.IQS_StreamInfo.PacketSamples];
FM_DemodParam_TypeDef FM_DemodParam;
Status = ASD_FMDemodulation(&Analog, &IQStream, reset, result, &FM_DemodParam);
delete[] result; ASD_Close(&Analog);
Status = Device_Close(&Device);
```

21.4 ASD_AMDemodulation

int ASD_AMDemodulation (void** AnalogMod, const IQStream_TypeDef* IQStreamIn, bool reset, float result[],AM_DemodParam_TypeDef* AM_DemodParam)	
功能描述	
对IQ数据进行AM解调。	
兼容性	0.55.0及之后版本支持
参数说明	
void** AnalogMod	运行Analog所需的内存空间引用。
const IQStream_TypeDef* IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
bool reset	重置缓存，对于一次连接解调来说，只有第一次调用函数的时候需要置成1，后面都是置0。
float result[]	AM解调后的信号数据，数据长度与IQ信号数据长度相同。

AM_DemodParam_TypeDef* AM_DemodParam	AM解调后返回的AM调制信号的AM调制速率、AM调制深度等参数。
double carrierOffsetHz	载波的频率偏移。
AM_DemodParam_TypeDef 详细定义	
double ModRate	AM调制速率。
double ModDepth	AM调制深度 或调制指数。
double ModRate_Avg	多次平均后的AM调制速率。
double ModDepth_Avg	多次平均后的AM调制深度 或调制指数。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	需要在ASD_Open之后进行调用。

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL; void* AlternIQStream = NULL; float ScaleToV = 0;
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);
IQS_TriggerInfo_TypeDef TriggerInfo;
MeasAuxInfo_TypeDef MeasAuxInfo;
IQStream_TypeDef IQStream;
Status = IQS_BusTriggerStart(&Device);
Status = IQS_GetIQStream_PM1(&Device, &IQStream);
Status = IQS_BusTriggerStop(&Device);
void* Analog = NULL; bool reset = 1; double carrierOffsetHz = 0;
ASD_Open(&Analog);
float* result = new float[IQStream.IQS_StreamInfo.PacketSamples];
AM_DemodParam_TypeDef AM_DemodParam;
IQS_GetIQStream(&Device, &AlternIQStream, &ScaleToV, &TriggerInfo, &MeasAuxInfo);
Status = ASD_AMDemodulation(&Analog, &IQStream, reset, result, &AM_DemodParam);
delete[] result; ASD_Close(&Analog);
Status = IQS_BusTriggerStop(&Device);
Status = Device_Close(&Device);
```

22 数字信号处理 DSP 迹线分析

22.1 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	
分析迹线的 IM3 参数。	
兼容性	0.55.0及之后版本支持
参数说明	
const double Freq_Hz []	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
const uint32_t TracePoints	迹线点数，即两个输入数组的长度。
TraceAnalysisResult_IP3_TypeDef * IM3Result	返回IP3的测量结果。
TraceAnalysisResult_IP3_TypeDef 详细定义	
double LowToneFreq	低音信号频率，单位随数据源。
double HighToneFreq	高音信号频率，单位随数据源。
double LowIM3PFreq	低频交调频率，单位随数据源。
double HighIM3PFreq	高频交调频率，单位随数据源。
float LowTonePower_dBm	低音功率, dBm。
float HighTonePower_dBm	高音功率, dBm。
float TonePowerDiff_dB	低音功率 - 高音功率。
float LowIM3P_dBc	LowIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - LowTonePower_dBm, 低频交调产物相对于最强主信号的强度。
float HighIM3P_dBc	HighIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - HighTonePower_dBm, 高频交调产物相对于最强主信号的强度。
float IP3_dBm	IP3的测试结果。
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; 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(), &MeasAuxInfo);
TraceAnalysisResult_IP3_TypeDef IM3Result;
Status = DSP_TraceAnalysis_IM3(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, &IM3Result);
Status = Device_Close(&Device);
```

22.2 DSP_TraceAnalysis_IM2

int DSP_TraceAnalysis_IM2(const double Freqs[], const float PowerSpec_dBm[], uint32_t TracePoints, TraceAnalysisResult_IP2_TypeDef* IM2Result)	
功能描述	
分析迹线的 IM2 参数。	
兼容性	0.55.0及之后版本支持
参数说明	
double Freqs[]	输入的频率数组。
float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
TraceAnalysisResult_IP2_TypeDef * IM2Result	返回IP2的测量结果。
TraceAnalysisResult_IP2_TypeDef 详细定义	
double LowToneFreq	低音信号频率，单位随数据源。
double HighToneFreq	高音信号频率，单位随数据源。
double IM2PFreq	低频交调频率，单位随数据源。
float LowTonePower_dBm	低音功率, dBm。

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

示例

```
int Status = -1; int DeviceNum = 0; void* Device = NULL; uint8_t IfDoConfig = 1;
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);
SWP_AutoSet(&Device, SWPChannelPowerMeas, &ProfileIn, &ProfileOut, &TraceInfo, IfDoConfig);
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(), &MeasAuxInfo);
TraceAnalysisResult_IP2_TypeDef IM2Result;
Status = DSP_TraceAnalysis_IM2(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, &IM2Result);
Status = Device_Close(&Device);
```

22.3 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
分析迹线的信道功率。

兼容性	0.55.0及之后版本支持
参数说明	
const double Freq_Hz[]	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
const uint32_t TracePoints	迹线点数，即两个输入数组的长度。
const double CenterFrequency	需要测量的信道中心频率。
const double AnalysisSpan	需要测量的信道带宽。
const double RBW	分辨率带宽。
DSP_ChannelPowerInfo_TypeDef * ChannelPowerResult	返回信道功率的测量结果。
DSP_ChannelPowerInfo_TypeDef 详细定义	
float ChannelPower_dBm	信道功率(dBm)。
float PowerDensity	信道功率密度(dBm/Hz)。
float ChannelPeakIndex	信道内的峰值索引。
double ChannelPeakFreq_Hz	信道内的峰值频率(Hz）。
float ChannelPeakPower_dBm	信道内的峰值功率(dBm）。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	无
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);
uint8_t IfDoConfig = 1;
SWP_AutoSet(&Device, SWPChannelPowerMeas, &ProfileIn, &ProfileOut, &TraceInfo, IfDoConfig);
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(), &MeasAuxInfo);
```

double CenterFrequency = 1e9; double AnalysisSpan = 50e6; DSP_ChannelPowerInfo_TypeDef ChannelPowerResult; double RBW ; Status = DSP_TraceAnalysis_ChannelPower(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTrace Points, CenterFrequency, AnalysisSpan, ProfileOut.RBW_Hz, &ChannelPowerResult); Status = Device_Close(&Device);

22.4 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	
分析迹线的 XdB带宽。	
兼容性	0.55.0及之后版本支持
参数说明	
const double Freq_Hz[]	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
const uint32_t TracePoints	迹线点数，即两个输入数组的长度。
const float XdB	峰值功率需要下降的dB。
TraceAnalysisResult_XdB_TypeDef* XdBResult	返回XdB的测量结果。
TraceAnalysisResult_XdB_TypeDef 详细定义	
double XdBBandWidth_Hz	XdB带宽(Hz)。
double CenterFreq_Hz	XdB带宽的中心频率(Hz)。
double StartFreq_Hz	XdB带宽的起始频率(Hz)。
double StopFreq_Hz	XdB带宽的终止频率(Hz)。
float StartPower_dBm	XdB带宽的起始频率对应的功率(dBm)。
float StopPower_dBm	XdB带宽的终止频率对应的功率(dBm)。
uint32_t PeakIndex	XdB带宽内的峰值索引。
double PeakFreq_Hz	XdB带宽内的峰值频率(Hz） 。
float PeakPower_dBm	XdB带宽内的峰值功率(dBm)。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	无
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(), &MeasAuxInfo);
TraceAnalysisResult_XdB_TypeDef XdBResult; float XdB = 3;
Status = DSP_TraceAnalysis_XdBBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
 XdB, &XdBResult);
Status = Device_Close(&Device);
```

22.5 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	
分析迹线的 占用带宽。	
兼容性	0.55.0及之后版本支持
参数说明	
const double Freq_Hz[]	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
uint32_t TracePoints	迹线点数，即两个输入数组的长度。
float OccupiedRatio	需要测试的占用带宽比例，通常情况下设置为0.99。
TraceAnalysisResult_OBW_TypeDef* OBWResult	返回占用带宽的测量结果。
TraceAnalysisResult_OBW_TypeDef 详细定义	
double OccupiedBandWidth	占用带宽(Hz)。
double CenterFreq_Hz	占用带宽的中心频率(Hz)。

double StartFreq_Hz	占用带宽的起始频率(Hz)。
double StopFreq_Hz	占用带宽的终止频率(Hz)。
float StartPower_dBm	占用带宽的起始频率对应的功率(dBm)。
float StopPower_dBm	占用带宽的终止频率对应的功率(dBm)。
float StartRatio	占用带宽的起始频率对应的功率占比。
float StopRatio	占用带宽的终止频率对应的功率占比。
uint32_t PeakIndex	占用带宽内的峰值索引。
double PeakFreq_Hz	占用带宽内的峰值频率(Hz）。
float PeakPower_dBm	占用带宽内的峰值功率(dBm)。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	无
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(), &MeasAuxInfo);
TraceAnalysisResult_OBW_TypeDef OBWResult; float OccupiedRatio = 0.99;
Status = DSP_TraceAnalysis_OBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, O
ccupiedRatio, &OBWResult);
Status = Device_Close(&Device);
```

22.6 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_TypeD
ef* ACPRResult)
```

功能描述	
分析迹线的 邻道功率比。	
兼容性	0.55.0及之后版本支持
参数说明	
const double Freq_Hz[]	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
const uint32_t TracePoints	迹线点数，即两个输入数组的长度。
const DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo	测试邻道功率比需要给定的变量。
TraceAnalysisResult_ACPR_TypeDef* ACPRResult	返回邻道功率比的测量结果。
DSP_ACPRFreqInfo_TypeDef 详细定义	
double RBW	分辨率带宽(Hz)。
double MainChCenterFreq_Hz	主信道中心频率(Hz)。
double MainChBW_Hz	主信道带宽(Hz)。
double AdjChSpace_Hz	邻道间隔，主信道中心频率与邻道中心频率的差值(Hz)。
uint32_t AdjChPair	邻道对。1代表左右2个邻道，2代表左右4个邻道。
TraceAnalysisResult_ACPR_TypeDef 详细定义	
float MainChPower_dBm	主信道功率(dBm)。
uint32_t MainChPeakIndex	主信道的峰值索引。
double MainChPeakFreq_Hz	主信道的峰值频率(Hz）。
float MainChPeakPower_dBm	主信道峰值功率(dBm)。
double L_AdjChCenterFreq_Hz	左邻道中心频率(Hz)。
double L_AdjChBW_Hz	左邻道带宽(Hz)。
float L_AdjChPower_dBm	左邻道功率(dBm)。
float L_AdjChPowerRatio	左邻道功率比（左邻道功率/主信道功率）。
float L_AdjChPowerDiff_dBc	左邻道功率差（左邻道功率-主信道功率 dBc）。
float L_AdjChPeakIndex	左邻道的峰值索引。
double L_AdjChPeakFreq_Hz	左信道的峰值频率(Hz）。
float L_AdjChPeakPower_dBm	左邻道的值功率(dBm)。
double R_AdjChCenterFreq_Hz	右邻道中心频率(Hz)。
double R_AdjChBW_Hz	右邻道带宽(Hz)。
float R_AdjChPower_dBm	右邻道功率(dBm)。

float R_AdjChPowerRatio	右邻道功率比（右邻道功率/主信道功率）。
float R_AdjChPowerDiff_dBc	右邻道功率差（右邻道功率-主信道功率 dBc）。
float R_AdjChPeakIndex	右邻道的峰值索引。
double R_AdjChPeakFreq_Hz	右信道的峰值频率(Hz）。
float R_AdjChPeakPower_dBm	右邻道的值功率(dBm)。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	无。
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(), &MeasAuxInfo);
DSP_ACPRFreqInfo_TypeDef ACPRFreqInfo;
ACPRFreqInfo.RBW = ProfileOut.RBW_Hz;
ACPRFreqInfo.MainChCenterFreq_Hz = 1e9;
ACPRFreqInfo.MainChBW_Hz = 50e6;
ACPRFreqInfo.AdjChSpace_Hz = 50e6;
ACPRFreqInfo.AdjChPair = 2;
TraceAnalysisResult_ACPR_TypeDef ACPRPowerInfo;
vector<TraceAnalysisResult_ACPR_TypeDef> ACPRResult(ACPRFreqInfo.AdjChPair);
Status = DSP_TraceAnalysis_ACPR(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints, A
CPRFreqInfo, ACPRResult.data());
Status = Device_Close(&Device);
```

22.7 DSP_SEMAnalysis

int DSP_SEMAnalysis(const DSP_SEMProfile_TypeDef* semProfile, const double Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, const double CenterFreq, DSP_SEMResult_TypeDef* SEMResult)	
功能描述	
对频谱数据执行发射模板分析。	
兼容性	0.55.62及之后版本支持
参数说明	
const DSP_SEMProfile_TypeDef* semProfile	SEM模版参考线配置。
const double Freq_Hz[]	输入的频率数组。
const float PowerSpec_dBm[]	输入的功率数组。
const uint32_t TracePoints	输入迹线点数。
const double CenterFreq	输入的中心频率。
DSP_SEMResult_TypeDef* SEMResult	返回SEM所有线段的测量结果。
DSP_SEMProfile_TypeDef 详细定义	
int mRefSetType	参考设置方式。 mRefSetType =0： 峰值参考； mRefSetType = 1：手动参考。
float mManualRefLevel	当mRefSetType = 1时，手动设置参考门限。
DSP_SEMSegmentProfile_TypeDef mSegments[16]	配置每一段参考线的开始/截止频率、开始/截止门限等，最多有16段线。
DSP_SEMSegmentProfile_TypeDef 详细定义	
bool mState	开启/关闭SEM功能 关闭=0； 使能 =1
double mStartFreq	开始频率。
double mLimitStart	开始门限。
double mStopFreq	截止频率。
double mLimitStop	截止门限。
int mMode	绝对 = 0; 相对 = 1。
int mPriority	要求 = 0； 建议 = 1。

DSP_SEMResult_TypeDef 详细定义	
DSP_SEMSegmentResult_TypeDef mSegmentResults[16]	返回SEM测量的结果。
DSP_SEMProfile_TypeDef mProfile	返回SEM测量时的配置。
float mRefLevel	返回SEM测量时采用的参考。
DSP_SEMSegmentResult_TypeDef 详细定义	
double mLowerFreq	低端频率。
float mLowerLevel	低端电平。
float mLowerMargin	低端余量。
bool mLowerPassOrFail	低端通过、失败。 0：通过；1：失败。
double mUpperFreq	高端频率。
float mUpperLevel	高端电平。
float mUpperMargin	高端余量。
bool mUpperPassOrFail	高端通过、失败。
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 = 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);
//参数配置可根据实际信号和频谱发射模板适当调整。
SWP_ProfileIn.FreqAssignment = CenterSpan;
SWP_ProfileIn.CenterFreq_Hz = 1e9;
SWP_ProfileIn.Span_Hz = 60e6;
SWP_ProfileIn.RefLevel_dBm = -20;
SWP_ProfileIn.RBWMode = RBW_Manual;
SWP_ProfileIn.RBW_Hz = 5e3;
SWP_ProfileIn.VBWMode = VBW_TenPercentRBW;
```

```
SWP_ProfileIn.Detector = Detector_Average;
SWP_ProfileIn.SpurRejection = Enhanced;
SWP_ProfileIn.SweepTimeMode = SWTMode_minSWTx20;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
std::vector<double >Frequency(TraceInfo.FullsweepTracePoints);
std::vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
while (1){
   MeasAuxInfo_TypeDef MeasAuxInfo;
   Status = SWP_GetFullSweep(&Device, Frequency.data(), PowerSpec_dBm.data(), &MeasAuxInfo);
   if (Status == APIRETVAL_NoError){
      // 以 802.11n（20MHz）频谱发射模板为例
      DSP_SEMProfile_TypeDef semProfile = { 0, 0, { {true, 9e6, 0, 11e6, -20, 1, 0}, {true, 11e6, -20, 20e6, -28, 1,
0}, {true, 20e6, -28, 30e6, -40, 1, 0} } };
      DSP_SEMResult_TypeDef result;
      Status = DSP_SEMAnalysis(&semProfile, Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTra
cePoints, SWP_ProfileOut.CenterFreq_Hz, &result);
       // i < 3 表示 semProfile 中 mSegments[16] 仅使能了 3 组数据。注意：0：通过；1：失败!!!
      for (int i = 0; i < 3; i++) {
         std::cout << "low: " << result.mSegmentResults->mLowerPassOrFail << ", high: " << result.mSegmentRes
ults->mUpperPassOrFail << std::endl;
      }
   }
}
Device_Close(&Device);/*关闭设备*/
```

23 数字信号处理 DSP 流数据的分析与处理

23.1 DSP_Open

int DSP_Open(void** DSP)	
功能描述	
打开DSP功能，并在内存中分配一定空间以存储DSP的相关数据。在调用DSP的其它函数之前必须先调用此函数。当需要同时调用某一条函数时，可通过改变多加入几个DSP指针来进行操作。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。调用此函数后，函数将返回当前打开的DSP功能的内存地址。后续在调用其它API时，必须通过此引用来索引此次的地址。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	必须在其它DSP函数调用前调用此函数，且只需在最开始前调用一次即可，后续其它函数可根据此函数返回的设备内存地址执行相关操作。对于任何非异常的DSP_Open调用，必须在整个功能使用需求结束之后，调用DSP_Close函数，以释放内存。
示例	请参考DSP_DDC_Execute()函数相关示例。

23.2 DSP_Close

int DSP_Close(void** DSP)	
功能描述	
关闭DSP功能，释放所开辟的内存空间。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
返回值	0：无异常；非0：异常，详见附录1。
调用约束	只需在程序执行的最后调用此函数，调用此函数后DSP功能关闭，内存空间释放，如需要重新调用DSP功能。则需要再次通过调用DSP_Open，打开DSP功能。

示例	请参考DSP_DDC_Execute()函数相关示例。

23.3 DSP_FFT_DeInit

int DSP_FFT_DeInit(DSP_FFT_TypeDef* IQToSpectrum)	
功能描述	
初始化配置FFT模式的相关参数。FFT模式下的FFT点数、窗型、抽取倍数等参数统一封装在DSP_FFT_TypeDef结构体中。	
兼容性	0.55.0及之后版本支持
参数说明	
DSP_FFT_TypeDef* IQToSpectrum	DSP_FFT配置结构体指针，为输入/输出变量。
DSP_FFT_TypeDef 详细定义	
uint32_t FFTSize	FFT分析点数。
uint32_t SamplePts	有效采样点数。
Window_TypeDef Window	指定FFT分析所使用的窗函数： FlatTop：具有良好的幅度准确度； Blackman_Nuttall：具有良好的频率分辨力； LowSideLobe：低副瓣窗。
TraceDetector_TypeDef TraceDetector	设置迹线检波器类型： 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。
uint32_t DetectionRatio	迹线检波比。
float Intercept	设置输出频谱截取，例如 Intercept = 0.8 则表示输出 80%的频谱结果。
bool Calibration	设置是否进行校准。
返回值	0：无异常；非0：异常，详见附录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。

23.4 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	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const DSP_FFT_TypeDef* ProfileIn	DSP_FFT配置结构体指针，为输入变量。 请参考DSP_FFT_DeInit ()函数同名结构体参数详细定义。
DSP_FFT_TypeDef* ProfileOut	DSP_FFT配置结构体指针，为输出变量。 请参考DSP_FFT_DeInit ()函数同名结构体参数详细定义。
uint32_t TracePoints	当前DSP_FFT配置下可获取到的频谱点数。
double *RBWRatio	返回RBW相对于采样率的比值。RBW = RBWRatio * IQSampleRate
返回值	0：无异常；非0：异常，详见附录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。

23.5 DSP_FFT_IQSToSpectrum

int DSP_FFT_IQSToSpectrum(void** DSP_FFT, const IQStream_TypeDef* IQStream, double Freq_Hz[], float PowerSpec_dBm[])	
功能描述	
将IQ数据转换成频谱数据，包括频率和功率。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const IQStream_TypeDef* IQStream	IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。

double Freq_Hz[]	返回频谱数据的频率数组。数组大小为TracePoints，由DSP_FFT_Configuration()函数输出得到。
float PowerSpec_dBm[]	返回频谱数据的功率数组。数组大小为TracePoints，由DSP_FFT_Configuration()函数输出得到。
返回值	0：无异常；非0：异常，详见附录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);
```

23.6 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	0.55.0及之后版本支持
参数说明	
DSP_DDC_TypeDef* DDC_ProfileIn	DSP_DDC配置结构体指针，为输入/输出变量。
DSP_DDC_TypeDef 详细定义	
double DDCOffsetFrequency	设置复混频的频率偏移值。
double SampleRate	设置复混频的采样率，需要与IQ数据采样率相同。
float DecimateFactor	设置重采样抽取倍数，范围:1 ~ 2^16。
uint64_t SamplePoints	设置复混频的采样点数。
返回值	0：无异常；非0：异常，详见附录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。

23.7 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	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const DSP_DDC_TypeDef* ProfileIn	DSP_DDC配置结构体指针，为输入变量。 请参考DSP_DDC_DeInit ()函数同名结构体参数详细定义。
DSP_DDC_TypeDef* ProfileOut	DSP_DDC配置结构体指针，为输出变量。 请参考DSP_DDC_DeInit ()函数同名结构体参数详细定义。
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。

23.8 DSP_DDC_Reset

void DSP_DDC_Reset(void** DSP)	
功能描述	
重置DDC中的缓存。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
返回值	无。
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。

23.9 DSP_DDC_GetDelay

void DSP_DDC_GetDelay (void** DSP, uint32_t* delay)	
功能描述	
获取DDC的延时。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
uint32_t* delay	返回抽取滤波器的延时点数。
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。

23.10 DSP_DDC_Execute

int DSP_DDC_Execute(void** DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)
功能描述
将IQ数据进行数字下变频。

兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const IQStream_TypeDef* IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
IQStream_TypeDef* IQStreamOut	输出IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
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);
//若每次做解调的 IQ 数据不是连续的，则需要先调用 DSP_DDC_Reset，再调用 DSP_DDC_Excute
DSP_DDC_Reset(&DSP);
Status = DSP_DDC_Execute(&DSP, &IQStream, &IQStreamOut);
```

Status = DSP_Close(&DSP); Status = Device_Close(&Device);

23.11 DSP_AudioAnalysis

void DSP_AudioAnalysis(const double Audio[], const uint64_t SamplePoints, const double SampleRate, DSP_AudioAnalysis_TypeDef* AudioAnalysis)	
功能描述	
分析音频的 音频电压(V)、音频频率(Hz)、信纳德(dB)和总谐波失真(%) 参数。	
兼容性	0.55.0及之后版本支持
参数说明	
const double Audio[]	音频信号数组。
const uint64_t SamplePoints	音频信号数组的长度。
const double SampleRate	音频信号的采样率。
DSP_AudioAnalysis_TypeDef* AudioAnalysis	返回音频分析的测量结果。
DSP_AudioAnalysis_TypeDef 详细定义	
double AudioVoltage	音频电压(V)。
double AudioFrequency	音频频率(Hz)。
double SINDA	信纳德(dB)。
double THD	总谐波失真(%)。
返回值	无。
调用约束	无。
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 = ASD_FMDemodulation(&AnalogMod, &IQStream, reset, result.data(), &FM_DemodParam);
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);
```

23.12 DSP_LPF_DeInit

void DSP_LPF_DeInit(Filter_TypeDef* LPF_ProfileIn)	
功能描述	
初始化LPF模式的相关参数。LPF模式下的滤波器抽头数、截止频率、阻带衰减等参数统一封装在Filter_TypeDef结构体中。	
兼容性	0.55.0及之后版本支持
参数说明	
Filter_TypeDef* LPF_ProfileIn	Filter_TypeDef配置结构体指针，为输入/输出变量。
Filter_TypeDef 详细定义	
int n	设置滤波器抽头数（n > 0）。
float fc	设置截止频率 (截止频率/采样率 0 < fc < 0.5)。
float As	设置阻带衰减（As > 0 , [dB]）。
float mu	设置分数采样偏移量（-0.5 < mu < 0.5）。
uint32_t SamplePts	设置采样点数（Samples > 0）。
返回值	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。

23.13 DSP_LPF_Configuration

void DSP_LPF_Configuration(void** DSP, const Filter_TypeDef* LPF_ProfileIn, Filter_TypeDef* LPF_ProfileOut)	
功能描述	
配置LPF模式的相关参数。LPF模式下的滤波器抽头数、截止频率、阻带衰减等参数统一封装在Filter_TypeDef结构体中。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const Filter_TypeDef* LPF_ProfileIn	Filter_TypeDef配置结构体指针，为输入/输出变量。 请参考DSP_LPF_DeInit ()函数同名结构体参数详细定义。
Filter_TypeDef* LPF_ProfileOut	Filter_TypeDef配置结构体指针，为输入/输出变量。 请参考DSP_LPF_DeInit ()函数同名结构体参数详细定义。
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。

23.14 DSP_LPF_Reset

void DSP_LPF_Reset(void** DSP)	
功能描述	
重置LPF中的缓存。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
返回值	无。
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。

23.15 DSP_LPF_Execute_Real

void DSP_LPF_Execute_Real(void** DSP, float Signal[], float LPF_Signal[])	
功能描述	
对实信号进行低通滤波。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
float Signal[]	输入的信号。
float LPF_Signal[]	经过低通滤波后的信号。
返回值	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;                    //设置幅值
NCO_Profile.Frequency = 60e3;                  //设置频率
NCO_Profile.Phase = 0;                         //设置相位
NCO_Profile.SampleRate = 100e3;                //设置采样率
NCO_Profile.Samples = 2000;                    //设置采样点数
vector<float> sin(NCO_Profile.Samples);        //存放产生的实信号
vector<float> LPF_Signal(NCO_Profile.Samples); //存放滤波后的信号
void* DSP = NULL;                  //开辟数字信号处理地址。
Status = DSP_Open(&DSP);           //开启信号处理功能。
Filter_TypeDef LPF_ProfileIn;      //配置需要生成滤波器系数的参数
Filter_TypeDef LPF_ProfileOut;     //输出生成滤波器系数的参数。
DSP_LPF_DeInit(&LPF_ProfileIn); //初始化配置LPF模式的相关参数。
LPF_ProfileIn.As = 90;  //配置滤波器的阻带衰减。
LPF_ProfileIn.fc = 0.25; //配置滤波器截止频率。
LPF_ProfileIn.mu = 0;  //配置分数采样偏移量。
LPF_ProfileIn.n = 90;   //配置滤波器阶数。
LPF_ProfileIn.Samples = NCO_Profile.Samples; //配置采样点数。
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileOut); //配置LPF的相关参数。
```

```
DSP_GenerateSineWaveform(sin.data(),&NCO_Profile);//生成实信号
DSP_LPF_Execute_Real(&DSP, sin.data(), LPF_Signal.data()); //低通滤波_实信号
Status = DSP_Close(&DSP);
```

23.16 DSP_LPF_Execute_Complex

void DSP_LPF_Execute_Complex(void** DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)	
功能描述	
对IQ信号进行低通滤波。	
兼容性	0.55.0及之后版本支持
参数说明	
void** DSP	运行DSP所需的内存空间引用。
const IQStream_TypeDef* IQStreamIn	输入IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
IQStream_TypeDef* IQStreamOut	输出IQ数据流的相关信息，包括IQ数据及相关配置信息。 请参考IQS_GetIQStream_PM1()函数同名结构体参数详细定义。
返回值	无。
调用约束	在DSP_LPF_Configuration后调用。
示例	

```
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;
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; IQStream_TypeDef IQStreamOut;
DSP_LPF_DeInit(&LPF_ProfileIn);
DSP_LPF_Configuration(&DSP, &LPF_ProfileIn, &LPF_ProfileIOut);
DSP_LPF_Reset(&DSP);
DSP_LPF_Execute_Complex(&DSP, &IQStream, &IQStreamOut);
Status = DSP_Close(&DSP);
Status = Device_Close(&Device);
```

24 附录 Appendix 1：API 返回值索引

表7　API 返回值说明

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

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

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