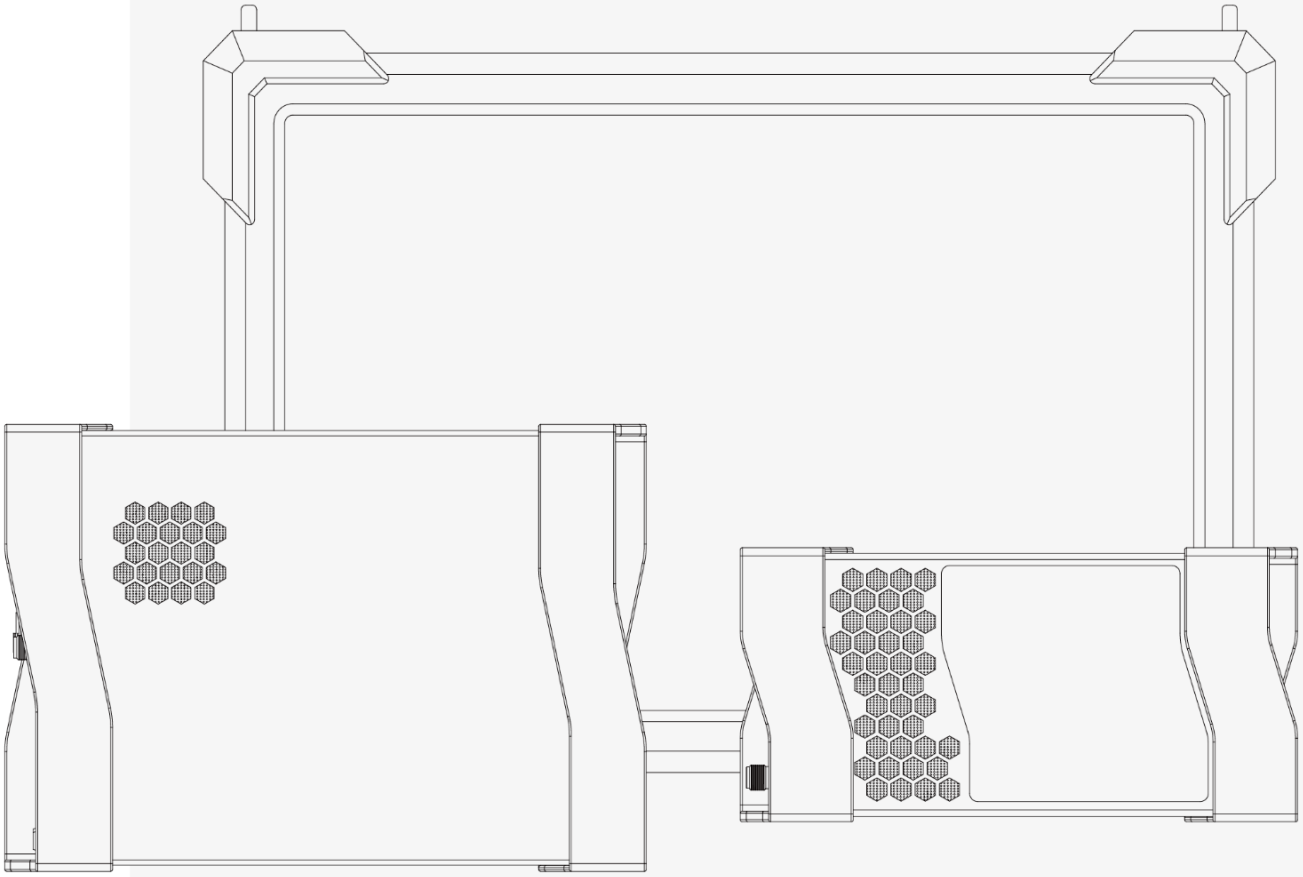
APPLICATION GUIDE

API Programming Guide



PX Series

SA Series

NX Series

V 0.55.63
2025-10-29

HAROGIC

Content

1 Version Information

Version	Content	Time
0.54.0	Initial Version 1. Device and API Version Introduction 2. Function Overview 3. API Usage Method 4. API Call Logic and Call Map 5. Important Variables and Setting Concepts 6. Usage and Parameter Explanation of Functions Related to Device and System, SWP, IQS, DET, RTA, MPS, ASG, DSP, ASD 7. Return Value Index Appendix	5-17-2023
0.55.27	1. Modified: Refactored the original Device chapter content. 2. Added: System Device and GNSS related functions chapter. 3. Added: The SWP_AutoSet function in SWP mode.	3-28-2024
0.55.61	1. Added: The Phase Noise Measurement Mode chapter. 2. Added: The IQS_GetIQStream_PM2 function in IQS mode. 3. Added: The ZeroSpan Mode chapter. 4. Added: The RTA_GetRealTimeSpectrum_Raw function in RTA mode. 5. Added: The Digital Demod chapter. 6. Added: The Pulse Det(Option) chapter. 7. Added: The description of the trace detector and detectors. 8. Deleted: The API usage method for Windows/Linux (already described in the API Usage Guide). 9. Deleted: The GetPatialUpdatedFullSweep function in SWP mode. 10. Modified: The Device and System chapter, adding Device_SetNetworkDeviceIP, Device_SetNetworkDeviceIP_PM1, and Device_GetFullUID. 11. Modified: All examples in version 0.55.27 have been replaced with the latest examples.	7-16-2025
0.55.62	1. Added: DSP adds a new function DSP_SEMAnalysis.	8-19-2025
0.55.63	1. Modification: Refactor AM/FM analog demodulation.	10-17-2025

2 Overview

This API system consists of dynamic linked libraries and header file. It is used for the secondary development of the real-time spectrum and receiver.

The measurement mode is a core concept of the API system. Different measurement modes have different test behavior and capability. As the first step of development, an appropriate measurement mode should be selected according to the task. The measurement modes of the the API system include the sweep mode (SWP), the IQ streaming mode (IQS), the power detection mode (DET), and the real time analysis mode (RTA). Fully understanding the mechanism of the measurement modes will help to maximize the devic's performance and obtain accurate measurement results.

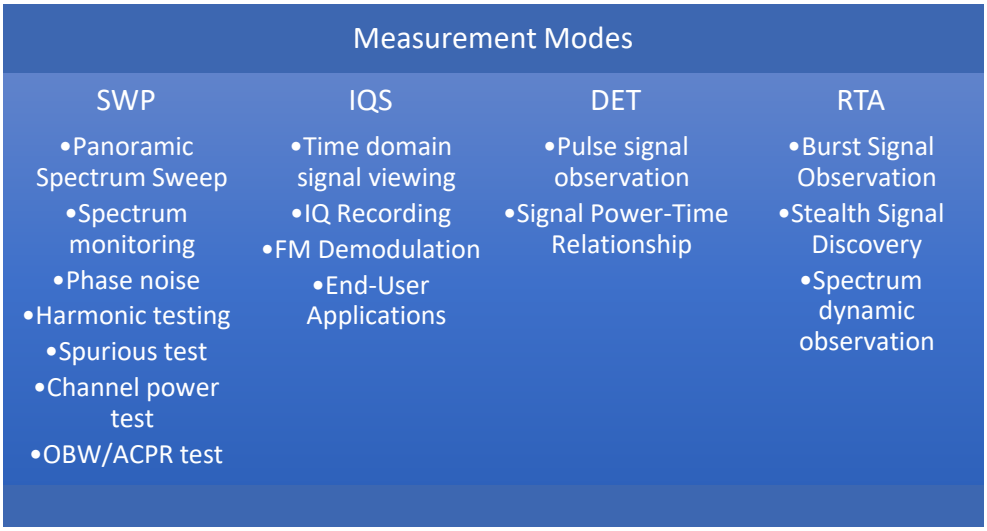


Figure 1 The Measurement Modes in API System and Main Application Scenarios

The basic flow of API calls consists of five steps: 1. open the device; 2. configurate the device to the specified measurement mode with appropriate parameter values; 3. obtain the measurement data; 4. execute user-defined process; 5. close the device.

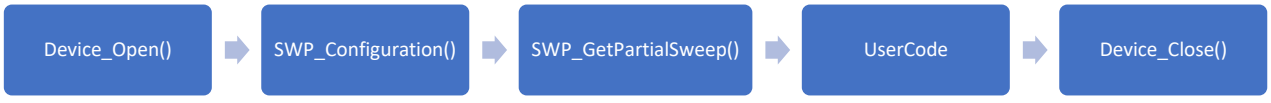


Figure 2 Typical Calling Steps for SWP Mode

Before starting your development, please read the chapter 5 API Call Logic and Call Map. This map can be the basic framework of your application and it is useful for building a robust and efficient application.

3 The Version of Device and API

The system is a joint operation of multiple softwares. In this system, the software involved includes 1) device master firmware (MFM), 2) FPGA firmware (FFM), 3) API, 4) SAStudio4; 5) other applications depend on user's requirements.

This system uses a unified software version naming rule to manage all the above software. We use the format of x.y.z to name software versions, where x is the primary version number, y is the secondary version number, and z is the sub-version number. The secondary version number y represents the compatibility mark of the software, and all software in the system must have the same y secondary version number to operate strictly and correctly.

For Example: MFM version = 0.55.1, FFM version = 0.55.2, API version = 0.55.4, SAStudio4 version = 1.55.45, this combination of all software with y = 55 will run matchingly.

For Example, if MFM version = 0.38.1, FFM version = 0.38.4, API version = 0.54.2, SAStudio4 version = 0.38.23, this combination where the API is an overrun version y = 54 and does not match the versions of MFM, FFM, SAStudio4, it will give an incorrect result.

Please note that the version of the API used corresponds to the version identified on the cover page of this brochure, so as to avoid any inconsistency between the description of this brochure and the actual API.

4 API Introduction of Function Categories

Table 1 Function Categories

Function Category	Description
Device and system	Global function, the functions under this category can be called in any measurement mode. including functions to turn on/off the device, set global settings, get device information and get device status.
SWP	In this mode, the receiver achieves the goal of frequency scanning performs via frequency hopping. Afterwards the baseband performs spectrum analysis on the time domain data within the analysis bandwidth acquired at each frequency point, and return the spectrum results to the user. SWP mode is suitable for frequency trace-oriented measurement and analysis applications. This category includes functions for configuration of SWP mode, acquisition of spectrum data, trigger control, etc.
IQS	In this mode, set the center frequency point to the specified frequency and keep the receiver state such as the local oscillation frequency fixed. The baseband acquires and Return time domain data within the analysis bandwidth to the user based on the specified trigger signal. IQS mode is suitable for signal recording, demodulation analysis, and simultaneous multi-dimensional analysis applications. This category includes functions for configuration of IQS mode, acquisition of spectrum data, and trigger control.
DET	In this mode, set the center frequency to the specified frequency and keeps the receiver state such as the local oscillation frequency fixed. The baseband performs (DET) on the time domain signal in the analysis bandwidth and return the power result to the user according to the specified trigger signal. DET mode is suitable for applications that are related with in-band power-time relationships, such as pulse parameter measurements. This category includes functions for configuring the DET mode, acquiring spectral data, and trigger control.
RTA	In this mode, the receiver sets the center frequency to the specified frequency and keeps the receiver state fixed such as the local oscillation frequency. The baseband performs continuous spectrum analysis of the time domain signal within the analysis bandwidth and return the spectrum results to the user. RTA mode is suitable for applications that focus on transient and burst signals, such as interference exclusion and identification of characteristic signals in complex electromagnetic environments. This category includes functions such as configuration of RTA mode, acquisition of spectrum data, and trigger control.

ASG	A global function that controls the device or one of the analog signal source options that can be called in any measurement mode. This category includes functions to set output mono, sweep, etc.
DSP	Generic post-processing functions, independent of hardware state. This category includes functions such as DDC, FFT analysis, and video detector for IQ data; and measurement and analysis of spectral traces, such as IM3, phase noise, channel power, and occupied bandwidth.
ASD	Analog demodulation class post-processing functions, independent of hardware state. This category includes functions such as AM demodulation, FM demodulation, etc.

5 API Call Logic and Call Map

5.1 API call map for standard sweep frequency analysis (SWP)

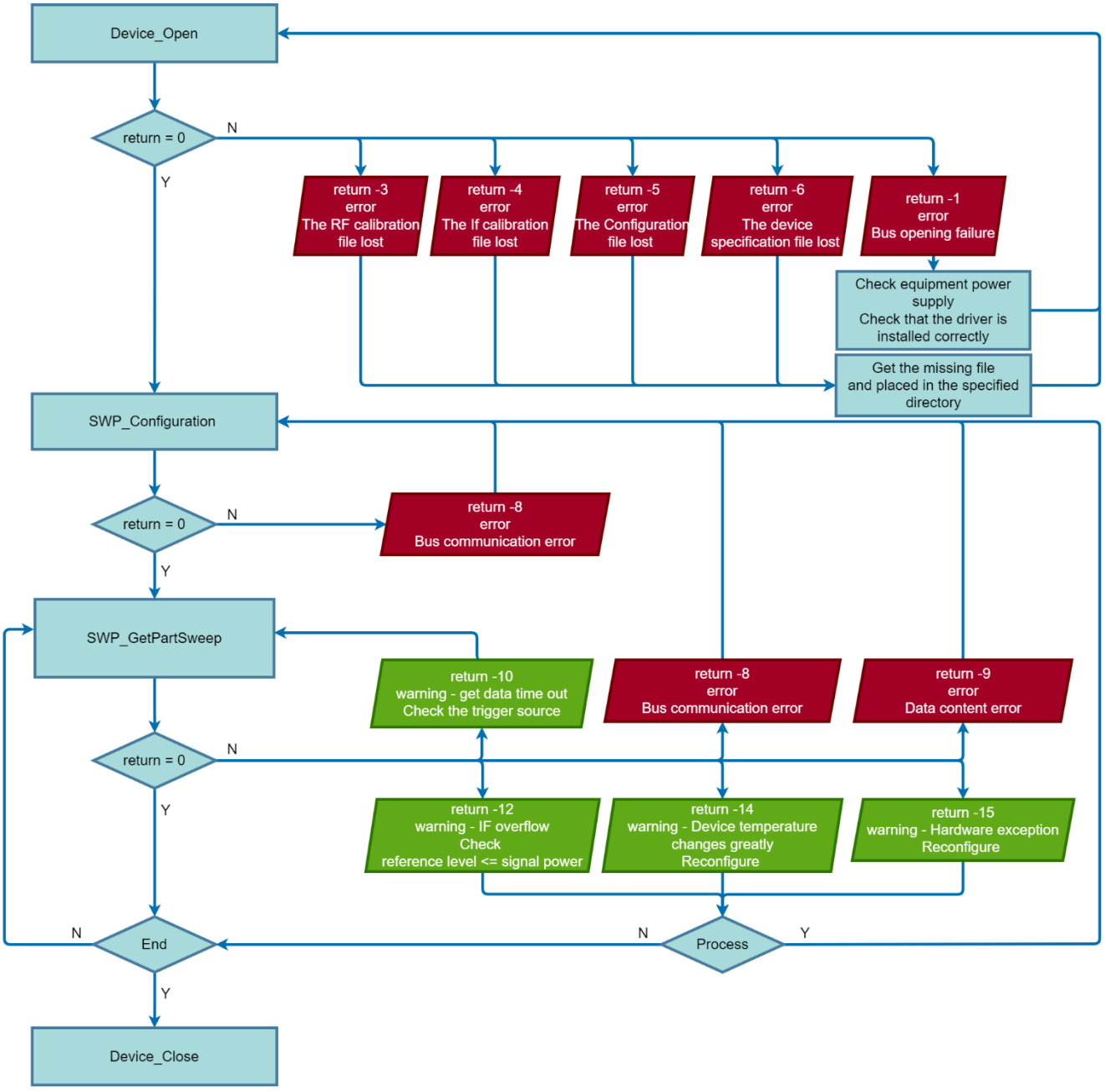


Figure 3 SWP Mode Call Flow Chart

5.2 API call map for IQ Streaming (IQS)

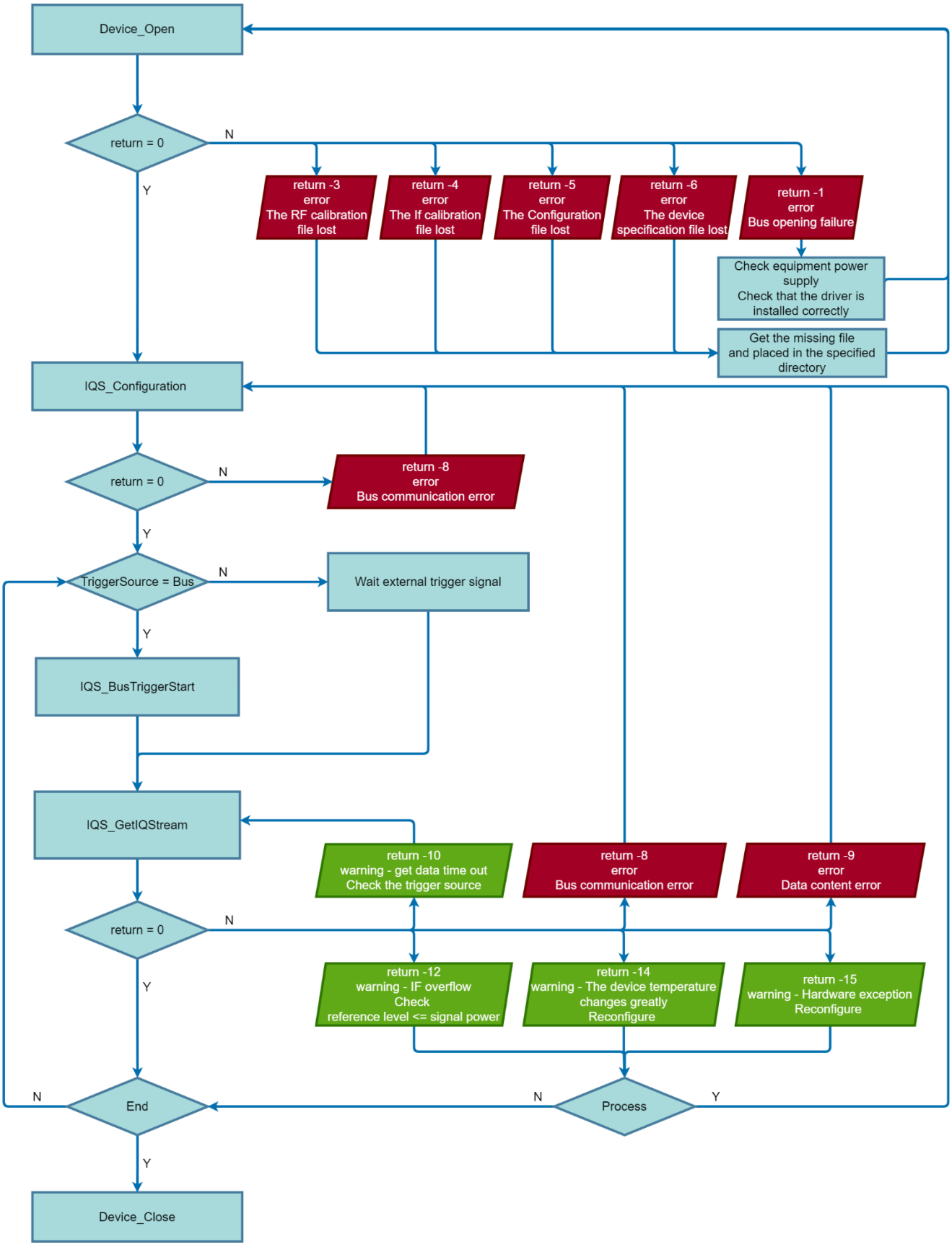


Figure 4 IQS mode Call Flow Chart (Trigger Mode is Fixed)

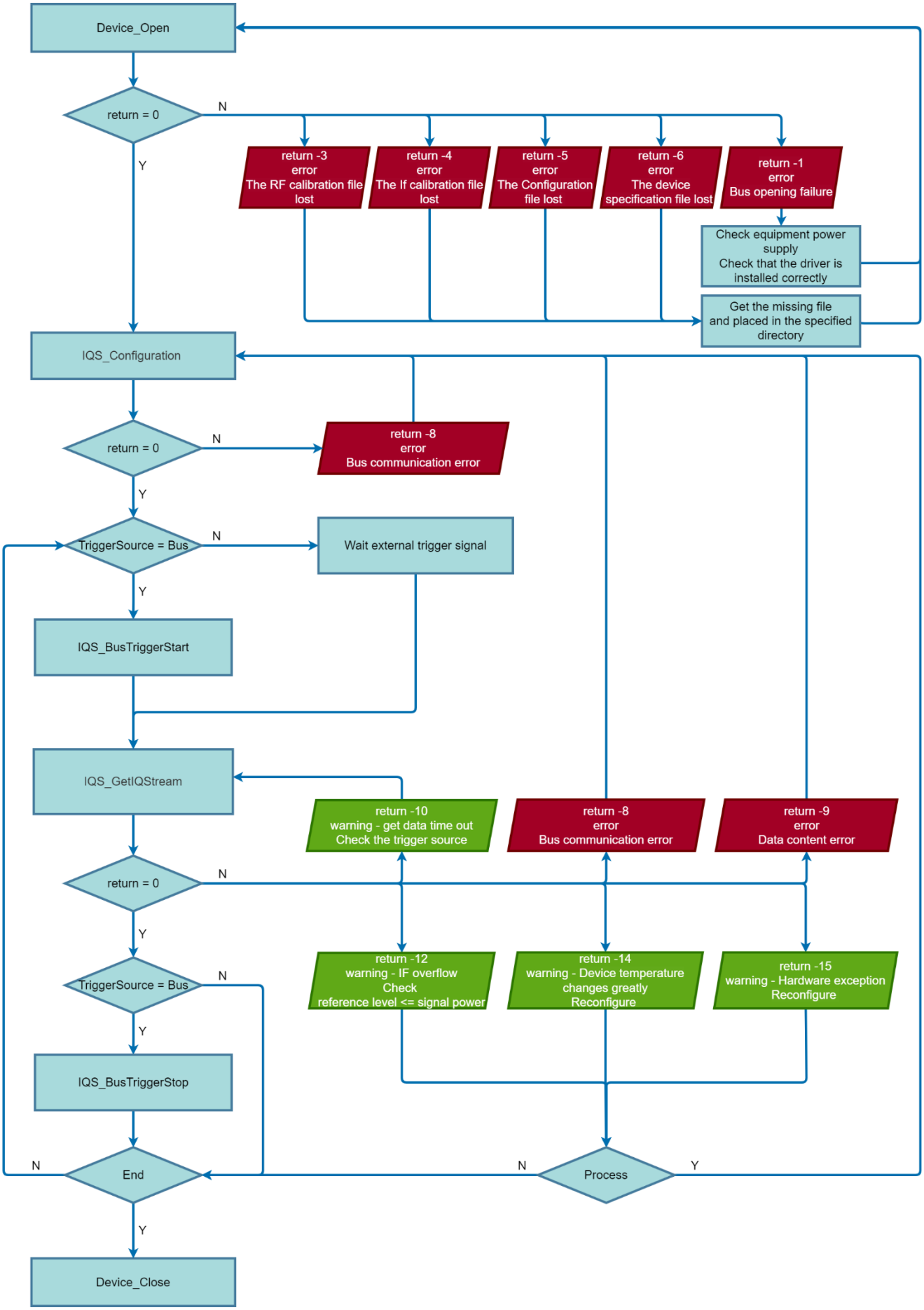


Figure 5 IQS Mode Call Flow Chart (Trigger Mode is Adaptive)

5.3 API call map for detection analysis (DET)

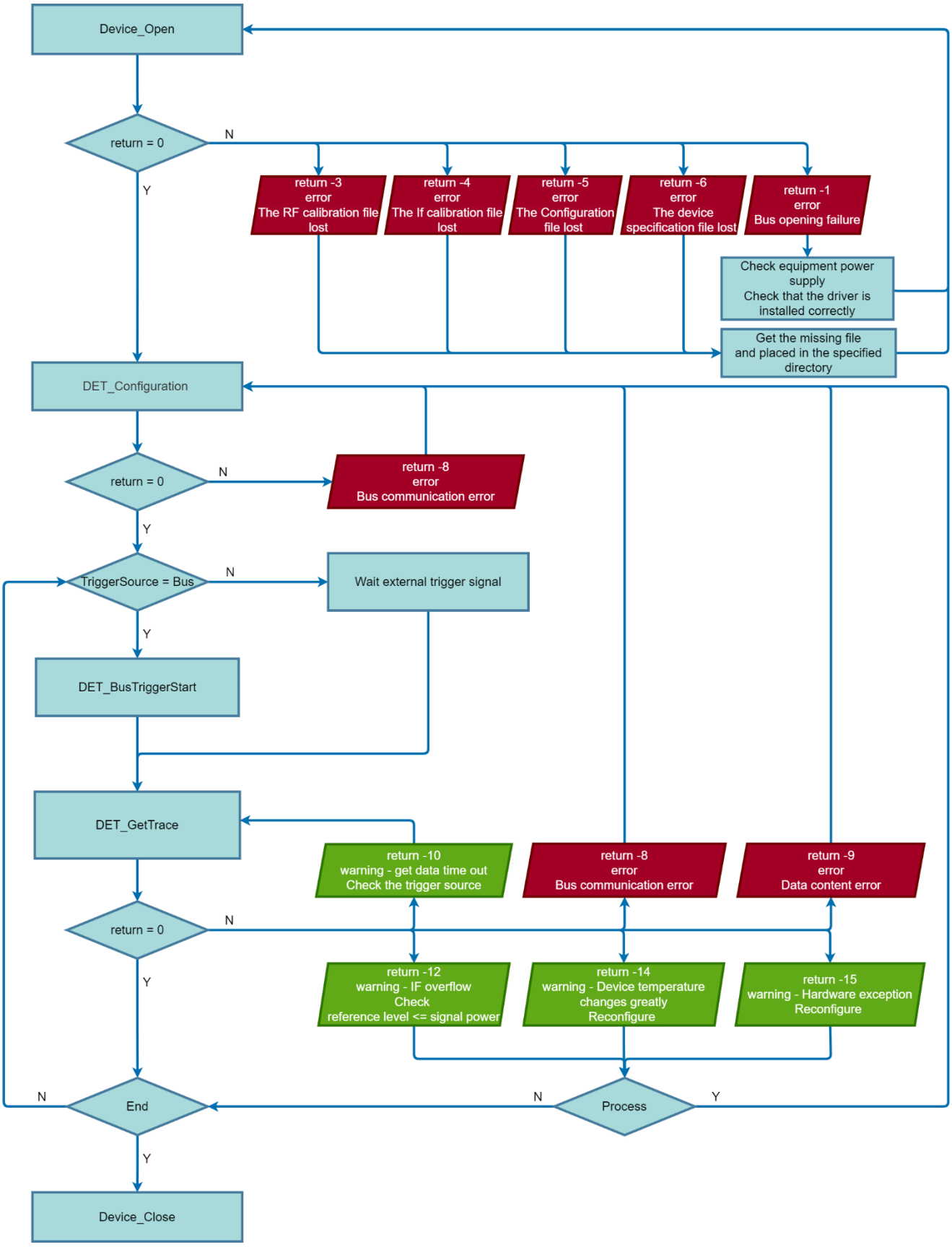


Figure 6 DET Mode Call Flow Chart (Trigger Mode is Fixed)

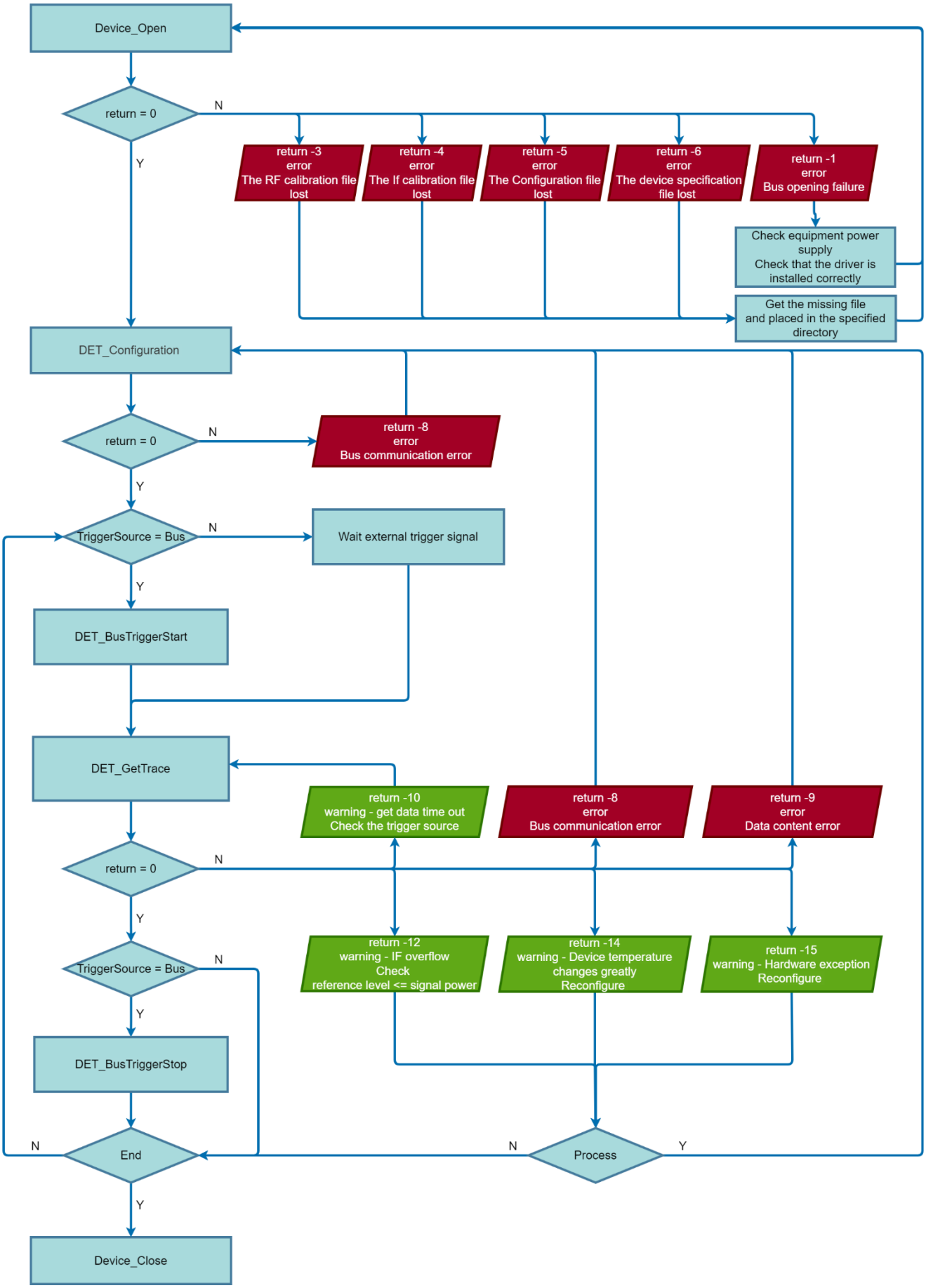


Figure 7 DET Mode Call Flow Chart (Trigger Mode is Adaptive)

5.4 API call map for Real-time Analysis (RTA)

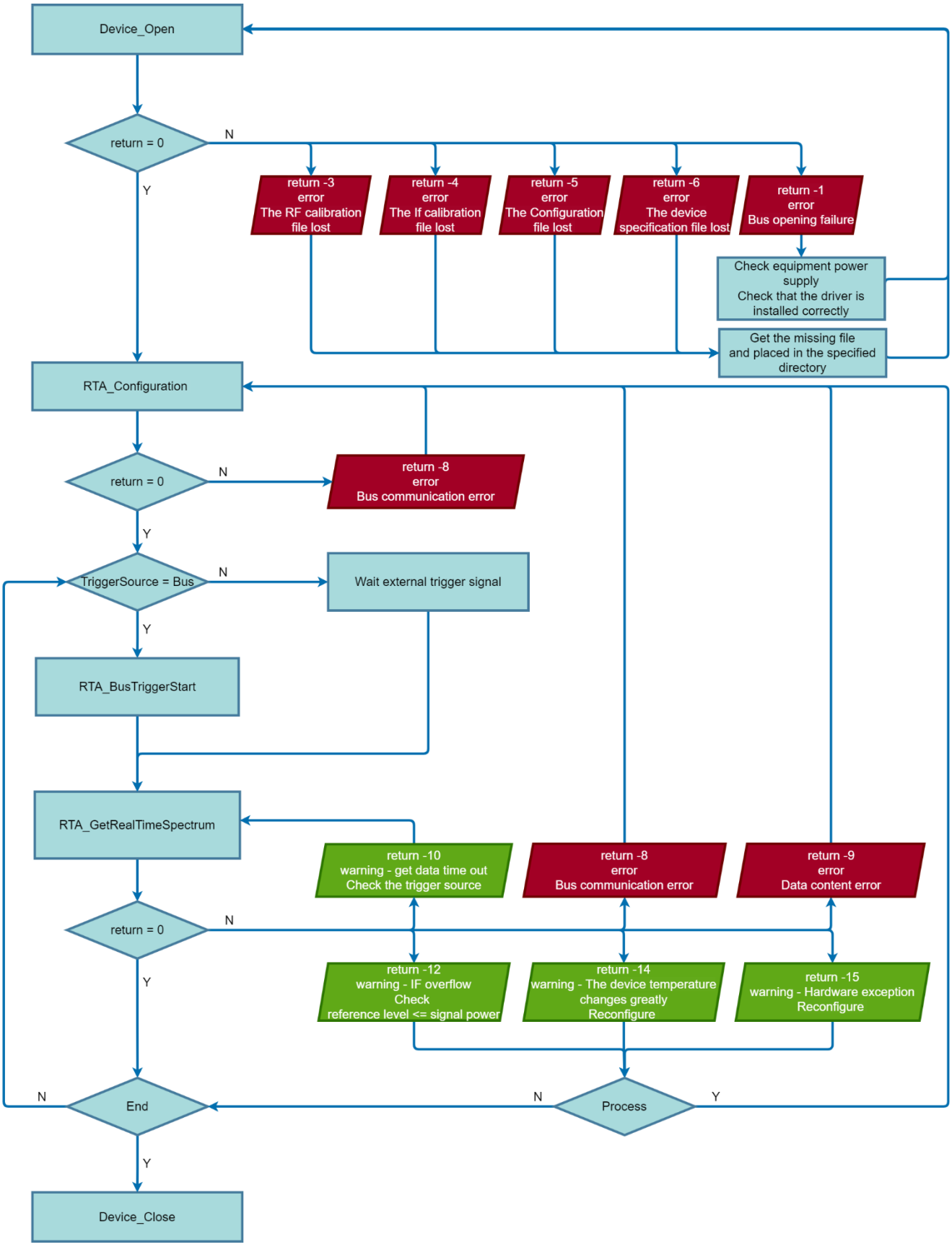


Figure 8 RTA Mode Call Flow Chart (Trigger Mode is Fixed)

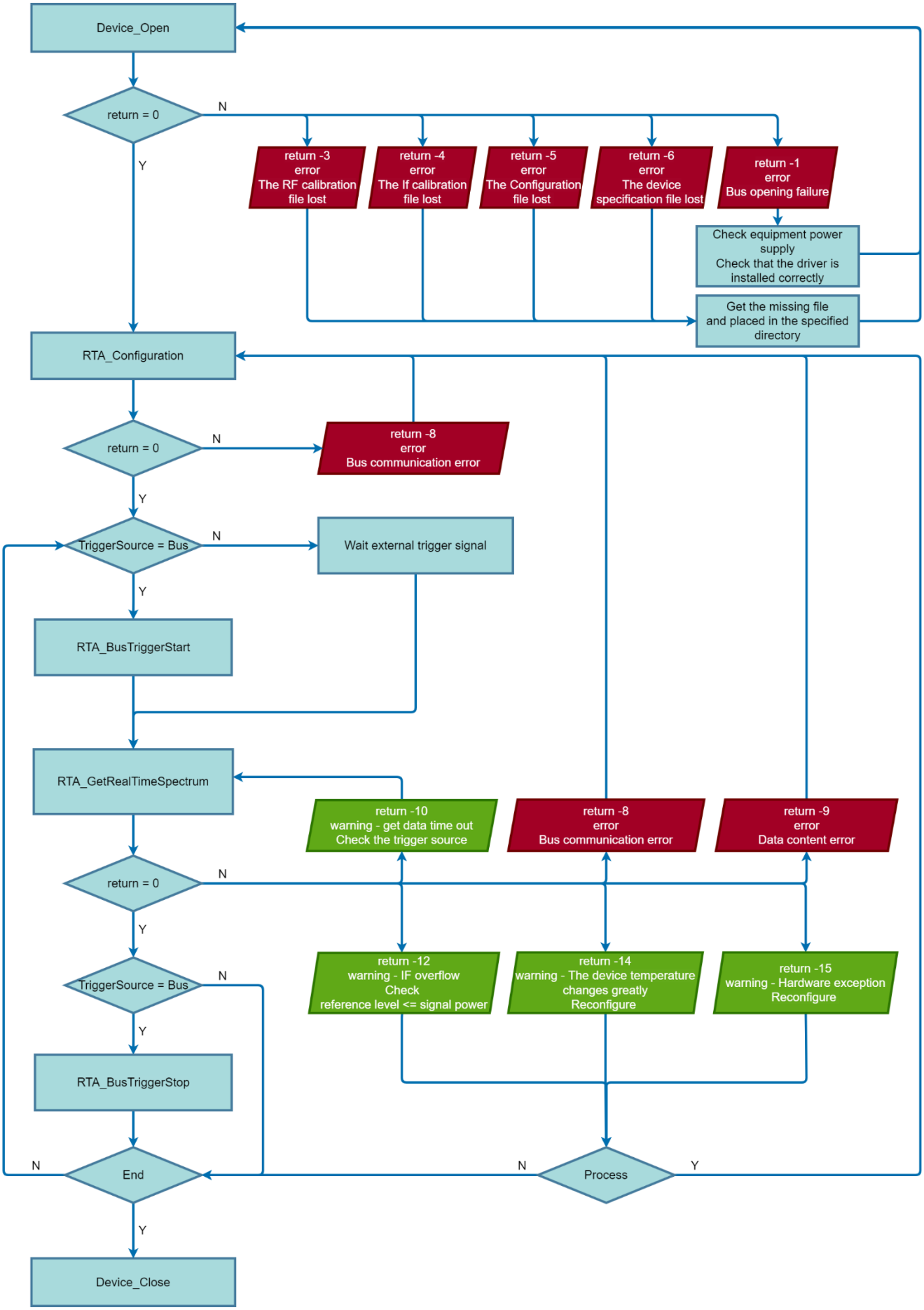


Figure 9 RTA Mode Call Flow Chart (Trigger Mode is Adaptive)

6 Important Variables Definition Reference

This section lists some of the important parameters involved in the spectrum analyzer/receiver devices. A good understanding of these parameters is important for proper use of the devices. They are summarized here for easy reference in case of preview or problems.

6.1 System

Table 2 System Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	Device memory pointer void** Device	Device/SWP/IQS/DET/ RTA	This parameter references the memory space required for device operation. When calling the API, you must use this reference to index the currently opened device.
2	DeviceUID	Device	Each device has a unique device ID, please use this ID to distinguish between different individual devices.

6.2 Amplititude

Table 3 Amplitude Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	RefLevel_dBm	SWP/IQS/DET/RTA	The system automatically configures the attenuator and preamplifier based on the reference level. The reference level can be understood as the maximum input power the system can handle without saturation. The system maintains a margin (typically 1-6 dB) when processing the reference level. Therefore, even if the input power exceeds the reference level at certain frequencies, the system may not trigger a saturation waring——this is normal behavior. For optimal dynamic range, set the reference level slightly higher than the expected maximum input power. For example, if the expected signal is a -3 dBm tone, setting the reference level to 0 dBm will provide good observation dynamics.
2	Atten	SWP/IQS/DET/RTA	The default attenuation mode is automatic (Atten = -1), where the system determines channel attenuation solely based on the reference level. To manually set channel attenutation, specify a desired value for Atten.
3	Preamplifier	SWP/IQS/DET/RTA	For devices equipped with a preamplifier,

			enabling/disabling it significantly impacts system noise performance and linearity: 1) Enabling the preamplifier: reduces system noise; decreases maximum linear input power and damage threshold; 2) Disabling the preamplifier: increases maximum tolerable input power; results in higher system noise. The system typically supports: 1) Automatic preamp control based on reference level; 2) Manual override to force-disable the preamp (preventing overload damage).
4	AnalogIFBW Grade	SWP/IQS/DET/RTA	For devices equipped with multiple analog IF filters, the system provides multiple IF channels with different characteristics. Each analog IF bandwidth option exhibits distinct performance in out-of-band rejection, in-band flatness, and group delay. Select the appropriate IF bandwidth setting based on application requirements.
5	IFGainGrade	SWP/IQS/DET/RTA	The system allows users to adjust IF gain to optimize spurious performance, linearity, and noise levels. Higher gain settings (indexed numerically) provide greater IF amplification, typically in 1 dB to 3dB increments per step. Total system gain = RF gain + IF gain. When maintaining a constant reference level (fixed total gain): 1) Increasing IF gain -> Reduces mixer input power -> Imporves spurious suppression and linearity -> Degrades noise performance; 2) Decreasing IF gain -> Increases mixer input power -> Worsens spurious/linearity -> Improves noise performance. Special cases: 1) At maximum RF gain(e.g., ref. level = -60 dBm): Further IF gain increase boosts total gain, potentially improving noise performance; 2) Below maximum RF gain(e.g., ref. level = 0 dBm): Higher IF -> Better spurious/linearity, worse noise; LowerIF gain -> Worse spurious/linearity, better noise.

6.3 Frequency

Table 4 Frequency Parameters and Related Concepts Information Explanation Table

Sequence	Parameters	Applicable Mode	Description
1	FreqAssignment	SWP	In SWP mode, this variable allows users to define the frequency scan range either in StartStop or CenterSpan format.
2	StartFreq_Hz StopFreq_Hz CenterFreq_Hz Span_Hz		
3	TracePointStrategy	SWP	In SWP mode, the spectrum analysis method is determined by TracePointStrategy: 1) When TracePointStrategy = BinSizeAssined: The system uses sweep-based analysis, where the trace point count is explicitly defined by TracePoints. In this mode, TraceAlign settings are ignored; 2) For other TracePointStrategy values: The system employs FFT-based analysis. Due to underlying software implementation and trace detection mechanisms, the native ananlysis frequencies cannot be perfectly aligned with the configured start/stop frequencies. The returned trace data points will slightly exceed the specified. User-adjustable handling methods: 1) Truncate endpoint data to match the desired frequency range; 2) Set TraceAlign = AlignToStart to force alignment at the start frequency (end frequency still requires truncation). Note for FFT mode: While users can specify a desired trace point count (TracePoints), hardware limitations typically prevent exact matches. The system returns the closest achievable point count.
4	TracePoints		
5	TraceAlign		

6.4 Analysis

Table 5 Analysis Parameters and Related Concepts Information Explanation Table

Sequence	Parameters and Concepts	Applicable mode	Description
1	SpurRejection	SWP	The system provides three spurious suppression modes: Off, Standard, and Enhanced. This feature effectively

			suppresses most composite spurious components but does not improve system residual responses. It also reduces sweep speed and time-varying signal measurement capability. 1) For steady-state signals (e.g., CW tones): Enabling spurious suppression significantly improves spurious-free dynamic range; 2) For fast time-varying signals (e.g., modulated signals): Activation may cause intermittent signal loss or inaccurate power measurements. Use with caution. Recommendation: Toggle the feature while observing spectral changes to determine if spurious suppression is suitable for your test scenario.
2	PowerBalance	SWP	In SWP mode, users can balance scan speed and power consumption by configuring the Power Consumption Balance parameter: 1) Power Consumption Blance = 0: The system operates at maximum sweep speed (highest power draw); 2) Power Consumption Balance = 40-1000 (typical range): Higher values reduce scan speed and lower power consumption. Critical Note: Higher Power Consumption Balance values significantly degrade time-varying signal detection capability. Use caution when testing highly dynamic signals.
3	Window	SWP/RTA	When performing FFT-based spectrum analysis, the system provides multiple window functions, each with distinct advantages. Please select the appropriate window based on your testing requirements: 1) FlatTop Window: Provides excellent amplitude accuracy; Significantly reduces amplitude errors caused by the picket-fence effect; Ideal for high-precision amplitude measurements; 2) Blackman-Nuttall Window: Features a narrow main lobe for high frequency resolution; Enables faster scanning speeds than FlatTop at the same RBW setting; Suitable for high-frequency-resolution and fast-sweep testing scenarios; 3) LowSideLobe Window: Features extremely low sidelobe levels, effectively suppressing interference from strong signals to adjacent frequencies. Ideal for test scenarios

			requiring high dynamic range or coexistence of strong and weak signals.
4	FFTExcutionStrategy	SWP	In standard spectrum analysis mode, users can select the signal processing method: 1) Auto mode: The system automatically selects FPGA or CPU processing based on RBW; 2) FPGA-Only: Significantly reduces CPU load; Slower sweep speeds ata RBW ⩽ 5KHz due to FFT size limitations; 3) CPU-Only: Supports FFT sizes > 64K points, enabling faster sweeps ata narrow RBW (⩽5KHz).
5	SweepTime	SWP/RTA	In this system, the sweep time is defined as the total time required to ccomplete one full scan from the start frequency to the stop frequency. When SweepTime = SWTMode_Manual, this parameter represents the absolute time; when *N is specified, this parameter is the scan time multiplier, i.e., scanning is performed at N times the minimum scan time.
6	DecimateFactor	IQS/DET/RTA	In IQS/DET/RTA mode, the system uses DecimateFactor to realize variable analysis bandwidth. Analysis bandwidth = Analysis bandwidth (DecimateFactor = 1) / DecimateFactor. Due to the limitations, the system will achieve the nearest available value for the DecimateFactor according to the desired DecimateFactor for configuration and feedback to the user.
7	BusTimeOut	IQS/DET/RTA	BusTimeOut sets an upper limit of execution time for functions related to fetching data, and the process will be ended if valid data cannot be fetched within that time so that the system does not wait indefinitely. In IQS/DET/RTA mode, this parameter needs to be set.

6.4.1 Detectors and Trace Detectors

Detector: Under the same local frequency point, the detector ratio frame data is collected, and according to the characteristics of the detector, the multi-frame data is detected frequency by frequency point, and the eigenvalue frame is finally generated. The following figure takes PoePeak Detector as an example to introduce the process of positive peak detection, where Frame 0, Frame 1 and Frame 2 are the data collected at different moments, and After PoePeak Detector is the data after positive peak detection.

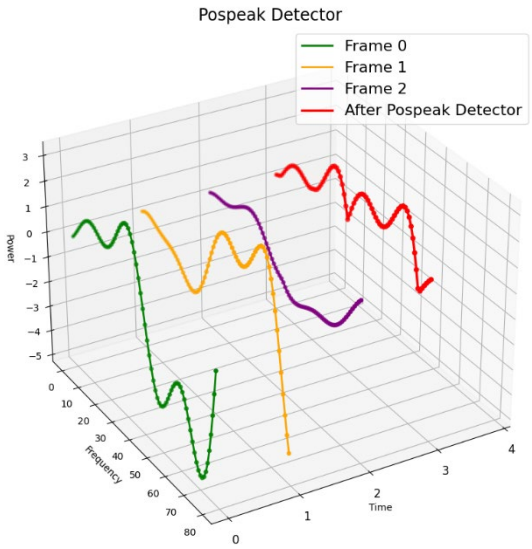


Figure 10 Positive peak detection

Trace Detector: According to the selected trace detector, the entire spectrum trace is detected in steps of trace detection ratio to generate the eigenvalue trace. The following figure takes PosPeak TraceDetector as an example to introduce the process of PosPeak Trace Detector, where Before PosPeak Trace Detector is the data before PosPeak Trace Detector and After PosPeak Trace Detector is the data after PosPeak Trace Detector.

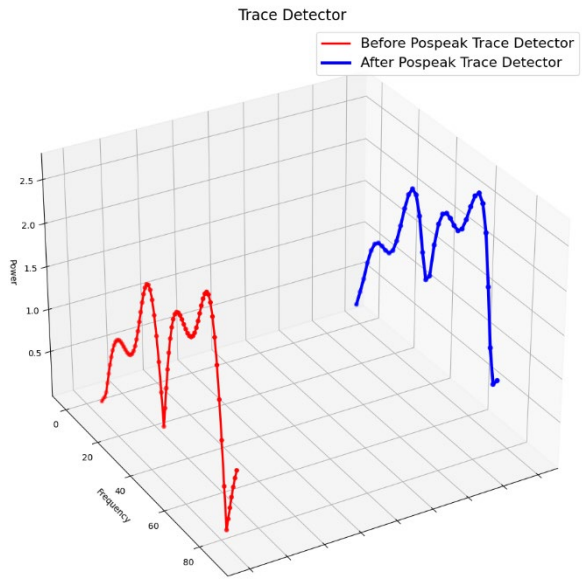


Figure 11 Positive peak trace detectors

6.5 Default Units

Table 6 Summary of Main Variables and Units

Variables	Unit
Frequency	Hz
Power	dBm
Voltage	V
Time	s

7 Device and System (main functions)

Before calling any API function associated with device hardware, you must call the Device_Open function to open the device. After completing your application's operations, call the Device_Close function to close the device and release memory space.

7.1 Device_Open

int Device_Open(void Device, int DeviceNum, const BootProfile_TypeDef* BootProfile, BootInfo_TypeDef* BootInfo)**	
Description	
The device must be opened before all future API calls. Calling this function to open a device and a handle will return. When there are multiple devices, open the devices separately by specifying different device number (DeviceNum) and the corresponding device handle can be obtained. Use the device handle to specify which device is to be manipulated in all the following API calls.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle. Use the device handle to specify which device is to be manipulated in the API calls.
int DeviceNum	Device Number. If there are multiple devices, you should open device with a different device number. The device number must accumulate from 0.
const BootProfile_TypeDef* BootProfile	Configurations for the device boot.
BootInfo_TypeDef* BootInfo	Feedback information of the device boot.
BootProfile_TypeDef	
PhysicalInterface_TypeDef PhysicalInterface	Specify the physical interface of the device. The interface must be set correctly, otherwise the driver cannot be opened. A device may be equipped with multiple interfaces. 1) USB: USB interface for data transfer; 2) ETH: 100M/1000M Ethernet interface for data transfer.
DevicePowerSupply_TypeDef DevicePowerSupply	Specify the power supply of the device. It must be set correctly or the device may not be opened. 1) USBPortAndPowerPort: USB data port and independent power port dual power supply; 2) USBPortOnly: only USB data port power; 3) Others: using a non-USB Bus such as the ETH.
IPVersion_TypeDef ETH_IPVersion	Ethernet IP version: IPv4.

uint8_t ETH_IPAddress[16]	When the physical interface is ETH, it is used to specify the IP address. For Example, the target IP address is 192.168.1.100, ETH_IPAddress[0] = 192; ETH_IPAddress[1] = 168; ETH_IPAddress[2] = 1; ETH_IPAddress[3] = 100; When the IPv4 is used, only the first 4 numbers need to be filled in, and the rest of the array does not need to be filled in.
uint16_t ETH_RemotePort	Specify the listening port if the physical interface is ETH.
int32_t ETH_ErrorCode	Return the error code of ETH connection if the physical interface is ETH.
int32_t ETH_ReadTimeOut	Specify the time out for data read if the physical interface is ETH. If the Device Configuration & Data functions do not respond within this time, the function will return an error code.
BootInfo_TypeDef	
DeviceInfo_TypeDef DeviceInfo	Device information including device UID, model, Hardware version, etc.
uint32_t BusSpeed	Data bandwidth of the data bus.
uint32_t BusVersion	Firmware version of the data bus.
uint32_t APIVersion	The version of the API under used. bit[31..16] represents the major version, bit[15..8] represents the minor version, bit[7..0] represents the revision.
int ErrorCodes[7]	Error code list.
int Errors	Error counts.
int WarningCodes[7]	Warning code list.
int Warnings	Warning counts.
DeviceInfo_TypeDef	
uint64_t DeviceUID	Unique ID of the device.
uint16_t Model	Device model.
uint16_t HardwareVersion	Version of the device hardware.
uint16_t MFWVersion	Version of the mcu firmware.
uint16_t FFWVersion	Version of the FPGA firmware.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	You must call this function before calling any other functions. It only needs to be called once at the very beginning. Subsequent functions can then perform related operations using the device memory address returned by this function. For every non-exceptional call to Device_Open, you must call the Device_Close function after the entire module's usage is complete to release the allocated memory.
Example	Please refer to the Device_QueryDeviceInfo_Realtime() function for related examples.

7.2 Device_Close

int Device_Close(void Device)**	
Description	
Shut down the device, and needs to be called to turn off the USB device and free up the memory space opened by the device in API calling.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Only need to call this function at the very end of program execution. After this function is called, the USB device connection will be closed and memory space will be released. If you need to use the device again, you'll need to re-establish the USB connection and open the device by calling Device_Open.
Example	Please refer to the Device_QueryDeviceInfo_Realtime() function for related examples.

7.3 Device_QueryDeviceState

int Device_QueryDeviceState(void Device, DeviceState_TypeDef* DeviceState)**	
Description	
Get the device status information of the current spectrometer, including device temperature, hardware working status, geographic time information (option support is required), etc. In a non-real-time way, it does not interrupt the data acquisition, but the information is only updated after acquiring the data packet.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DeviceState_TypeDef* DeviceState	Structure pointer to information about the current device state. The information update to the latest value after calling this function.
DeviceState_TypeDef	
int16_t Temperature	The temperature of the device. Degree Celsius = 0.01 * Temperature.
double AbsoluteTimeStamp	The absolute time stamp provided by the insystem GNSS.
float Latitude	Latitude: positive for north, negative for south, thus distinguishing north from south latitude.
float Longitude	Longitude: positive for east, negative for west, thus distinguishing east from west longitude.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.

Calling constraints	Must be called after Device_Open.
Example	Please refer to the Device_QueryDeviceInfo_Realtime() function for related examples.

7.4 Device_QueryDeviceState_Realtime

int Device_QueryDeviceState_Realtime(void Device, DeviceState_TypeDef* DeviceState)**	
Description	
This function gets device status in real-time, including device temperature, hardware operational status, and geographical time information (requires optional support). It occupies the data channel for short periods during acquisition.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DeviceState_TypeDef* DeviceState	Refer to the detailed definition of the structure parameter used in the Device_QueryDeviceState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the Device_QueryDeviceInfo_Realtime() function for related examples.

7.5 Device_QueryDeviceInfo

int Device_QueryDeviceInfo(void Device, DeviceInfo_TypeDef* DeviceInfo)**	
Description	
This function retrieves device information, including the device serial number and software/hardware versions. It operates in a non-real-time manner and doesn't interrupt data acquisition, but the information is only updated after a data packet is received.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DeviceInfo_TypeDef* DeviceInfo	A pointer to a structure that returns information about the current device serial number and hardware and software versions.When this function is called, the information pointed to by this pointer will be updated to the latest values. Refer to the detailed definition of the structure parameter used in the Device_Open() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the Device_QueryDeviceInfo_Realtime() function for related examples.

7.6 Device_QueryDeviceInfo_Realtime

int Device_QueryDeviceInfo_Realtime(void Device, DeviceInfo_TypeDef* DeviceInfo)**	
Description	
This function gets device information, including the device serial number and software/hardware version numbers. It's a real-time acquisition that temporarily occupies the data channel, but it guarantees you'll always get immediate device information.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DeviceInfo_TypeDef* DeviceInfo	A pointer to a structure that returns information about the current device serial number and hardware and software versions.When this function is called, the information pointed to by this pointer is updated to the latest values. Refer to the detailed definition of the structure parameter used in the Device_Open() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
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 Device and System (the rest)

8.1 Device_SetSysPowerState

int Device_SetSysPowerState(void Device, SysPowerState_TypeDef SysPowerState)**	
Description	
Set the power state of the device, including power-on operation, RF partial power-down, RF partial standby, and so on.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
SysPowerState_TypeDef SysPowerState	Device power consumption control: 1) PowerOn: power on all the parts of the device; 2) RFPowerOff: power down the RF part of the device.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
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 SetFanState_TypeDef SetFanState, const float ThreshouldTemperature)**	
Description	
Controls the device fan operating mode.(Supported only by N45, N60, M60, M80 devices)	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const SetFanState_TypeDef	Fan operating mode:

SetFanState	1) FAN_FORCED_ON: the system fan is forced on; 2) FAN_FORCED_OFF: the system fan is forced off; 3) FAN_AUTO: automatic mode.
const float ThreshouldTemperature	Threshold temperature for fan auto control. When the FanState is set to FAN_AUTO, the system fan will be automatically turned on if the threshold temperature is reached. The system fan will also be turned off if temperature is 10 celsius below the threshold.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
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)**	
Description	
Calibrate the reference clock frequency by the 1PPS of insystem GNSS or the periodic external trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
ClkCalibrationSource_TypeDef ClkCalibrationSource	Timing signal source used for calibration: 1) CalibrateByExternal: calibrate by the periodic external trigger; 2) CalibrateByGNSS1PPS: calibrate by the 1PPS of insystem GNSS.
const double TriggerPeriod_s	The period of periodic external trigger. The accuracy of the period will determin the calibration effect.
const uint64_t TriggerCount	Specify the number of triggers to be used for calibration. For scenarios where GNSS1PPS is used for calibration, the more triggers, the better the

	jitter error suppression and the better the calibration, but the longer the calibration time.When using GNSS1PSS, it is recommended that the number of triggers is greater than 30, i.e., the calibration takes more than 30 seconds.
const bool RewriteRFCal	1) 0：the calibration result is not written into the calibration file, and the calibration result is invalid after the device is powered down; 2) 1：the calibration results are written to a calibration file and remain calibrated after the device is powered on.
double* RefCLKFreq_Hz	Feedback the new reference clock frequency obtained from this calibration.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	
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])	
Description	
When using devices with ETH interfaces, obtain information such as IP addresses and subnet masks for all devices in the network.	
Compatibility	0.55.0 and later.
Parameter description	
uint8_t* DeviceCount	Device count.
NetworkDeviceInfo_TypeDef DeviceInfo[64]	Return device information, including device serial number, device type, and device version.

uint8_t LocalIP[4]	Return local IP address.
uint8_t LocalMask[4]	Return local subnet mask.
NetworkDeviceInfo_TypeDef	
uint64_t DeviceUID	Device Unique ID.
uint16_t Model	Device Model.
uint16_t HardwareVersion	Device Hardware Version.
uint32_t MFWVersion	Device Main Control Firmware Version.
uint32_t FFWVersion	Device FPGA Firmware Version.
uint8_t IPAddress[4]	IP address.
uint8_t SubnetMask[4]	Subnet mask.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None
Example	
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])	
Description	
When using devices with ETH interfaces, obtain information such as IP addresses and subnet masks for all devices in the network.	
Compatibility	0.55.0 and later.
Parameter description	
const uint64_t DeviceUID	Device Unique ID.
const uint8_t IPAddress[4]	Enter the IP address to configure.
const uint8_t SubnetMask[4]	Enter the subnet mask to configure.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
int Status = -1; void* Device = NULL; uint64_t DeviceUID = 0x31325119004c0048; **IPVersion_TypeDef ETH_IPVersion = IPv4;** **uint8_t IPAddress[4] = { 192,168,2,100 }; uint8_t SubnetMask[4] = { 255, 255, 255,0 };** **uint8_t ETH_IPAddress[4] = { 192,168,2,101 };** **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])	
Description	
Set a new IP address and subnet mask for the device through the existing IP address of the NX Series device.	
Compatibility	0.55.0 and later.
Parameter description	
const uint8_t DeviceIP[4]	Enter the current IP address.
const uint8_t IPAddress[4]	Enter the IP address to configure.
const uint8_t SubnetMask[4]	Enter the subnet mask to configure.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None
Example	
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)	
Description	
Get full UID information.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
uint64_t* UID_L64	The lower 64 bits of the device's corresponding UID.
uint32_t* UID_H32	The upper 32 bits of the device's corresponding UID.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
Get hardware status information.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
HardWareState_TypeDef* HardWareState	Return information such as the GNSS peripheral type, whether the device supports signal source functionality, and whether the device supports variable ADC sample rates.
HardWareState_TypeDef	
GNSSPeriphType_TypeDef GNSSPeriphType	GNSS Peripheral Type: 1) GNSS_None = 0: No peripherals; 2) GNSS_For_EIO = 1: EIO; 3) GNSS_For_NX = 2: NX; 4) GNSS_For_PX = 3: PX.
GNSSType_TypeDef GNSSType	GNSS receiver type: 1) None_GPS = 0: No GPS receiver; 2) GNSS_GPS = 1: Standard GPS; 3) GNSS_GPS_Pro = 2: High-performance GPS.
OCXOType_TypeDef OCXOType	OCXO type on GNSS: 1) None_OCXO = 0: Without OCXO; 2) GNSS_OCXO = 1: Ordinary OCXO on GNSS; 3) GNSS_DOCXO = 2: Disciplined OCXO on GNSS.
uint8_t InternalOCXO	Whether the internal reference clock of the device is a constant temperature crystal oscillator.
uint8_t SignalSourceEn	Whether the device supports the signal source function.
uint8_t ADC_VariableRateEn	Whether the device supports ADC variable sampling rate.
uint8_t IM3_filter	Complement IF filter (IM3 enhanced).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.

Example
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)	
Description	
Query device information by device number.	
Compatibility	0.55.0 and later.
Parameter description	
int DeviceNum	Specifies the device number that can be used to select a device when more than one device is connected to the host, and the number is accumulated from 0.
const BootProfile_TypeDef* BootProfile	Set up the startup configuration.
BootInfo_TypeDef* BootInfo	Feedback information of the device boot.
BootProfile_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_Open() function.
BootInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_Open() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	This function is called to ensure that the device is not open, that is, before Device_Open.
Example	
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)**	
Description	
Configure the center frequency scan parameters.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
double StartFreq_Hz	Start frequency in Hz.
double StopFreq_Hz	Stop frequency in Hz.
uint16_t SweepPts	The number of frequency points to scan.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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 System Device and GNSS Related Functions

9.1 Device_SetGNSSAntennaState

int Device_SetGNSSAntennaState(void Device, const GNSSAntennaState_TypeDef GNSSAntennaState)**	
Description	
This function is called to set the GNSS antenna status when using the GNSS function.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const GNSSAntennaState_TypeDef GNSSAntennaState	Set GNSS antenna status: Refer to the detailed definition of the structure parameter used in the Device_GetGNSSAntennaState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the Device_GetGNSSAntennaState() function for related examples.

9.2 Device_GetGNSSAntennaState

int Device_GetGNSSAntennaState(void Device, GNSSAntennaState_TypeDef* GNSSAntennaState)**	
Description	
When using the GNSS function, this function can be called to obtain the GNSS antenna status in a non-real-time manner (optional feature required), meaning it does not interrupt data acquisition, but the information is only updated after a data packet is retrieved.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSSAntennaState_TypeDef* GNSSAntennaState	Set GNSS antenna status: 1) GNSS_AntennaExternal: External antenna; 2) GNSS_AntennaInternal: Internal antenna.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
When using the GNSS function, this function can be called to obtain the GNSS antenna status in a real-time manner (optional support is required), but the real-time mode will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSSAntennaState_TypeDef* GNSSAntennaState	Refer to the detailed definition of the structure parameter used in the Device_GetGNSSAntennaState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	

```
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)**

Description	
When using the GNSS function, call this function to acquire elevation information about the location of the GNSS antenna in a non-real-time manner.(Option support required), i.e., the data acquisition is not interrupted, but the information is only updated after the packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
int16_t* Altitude	Return the elevation of the GNSS position.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be calledafter Device_Open.
Example	
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)	
Description	
Call this function to parse GNSS time and date information. (Requires optional support)	
Compatibility	0.55.0 and later.
Parameter description	
double ABSTimestamp	Return the absolute timestamp corresponding to the current packet.
int16_t* hour	Return the hour in the GNSS time and date information.
int16_t* minute	Return the minute in the GNSS time and date information.
int16_t* second	Return the second in the GNSS time and date information.
int16_t* Year	Return the year in the GNSS time and date information.
int16_t* month	Return the month in the GNSS time and date information.
int16_t* day	Return the day in the GNSS time and date information.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	

```
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)**	
Description	
This function is called to set the DOCXO working state when using GNSS functionality.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const DOCXOWorkMode_TypeDef DOCXOWorkMode	Set DOCXO antenna status: 1) DOCXO_LockMode: disciplined mode; 2) DOCXO_HoldMode: track mode.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the Device_GetDOCXOWorkMode_Realtime() function for related examples.

9.7 Device_GetDOCXOWorkMode

int Device_GetDOCXOWorkMode(void Device, DOCXOWorkMode_TypeDef* DOCXOWorkMode)**	
Description	
When using GNSS functionality, this function can be called to obtain the GNSS DOCXO working status in a non-real-time manner (optional support is required), without interrupting the data acquisition, but the information is only updated after the packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DOCXOWorkMode_TypeDef*	Set DOCXO antenna status:

DOCXOWorkMode	Refer to the detailed definition of the structure parameter used in the Device_SetDOCXOWorkMode() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
When using GNSS functions, calling this function can get the GNSS DOCXO working status in real time (optional support is required), but the real time mode will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DOCXOWorkMode_TypeDef* DOCXOWorkMode	Refer to the detailed definition of the structure parameter used in the Device_SetDOCXOWorkMode() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
When using GNSS functionality, this function can be called to obtain GNSS device status in a non-real-time manner (optional support is required). The non-real-time mode does not interrupt the data acquisition, but the information is only updated after the packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSSInfo_TypeDef* GNSSInfo	GNSS get device information.
GNSSInfo_TypeDef	
float latitude	Return latitude of GNSS antenna.
float longitude	Return longitude of GNSS antenna.
int16_t altitude	Return altitude of GNSS antenna.
uint8_t SatsNum	Return number of satellites currently in use of GNSS antenna.
uint8_t GNSS_LockState	Return GPS locked state.
uint8_t DOCXO_LockState	Return GDOCXO locked state.
DOCXOWorkMode_TypeDef **DOCXO_WorkMode**	Return DOCXO working state. Refer to the detailed definition of the structure parameter used in the Device_SetDOCXOWorkMode() function.
GNSSAntennaState_TypeDef **GNSSAntennaState**	Return antenna status. Refer to the detailed definition of the structure parameter used in the Device_GetGNSSAntennaState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
When using GNSS functionality, this function can be called to get GNSS device status in a real-time manner (optional support is required). The real-time mode is acquired in real time, but it will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSSInfo_TypeDef* GNSSInfo	GNSS gets the device information.
GNSSInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_GetGNSSInfo() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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)**	
Description	
GNSS signal-to-noise ratio (optional)), non-real-time mode, without interrupting the data acquisition, but the information is only updated after the packet is acquired.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSS_SatDate_TypeDef*	Return GNSS signal-to-noise ratio information.

GNSS_SatDate	
GNSS_SatDate_TypeDef	
uint8_t SatsNum_All	Current range visible satellite.
uint8_t SatsNum_Use	Number of satellites used for positioning.
GNSS_SNR_TypeDef **GNSS_SNR_UsePos**	Satellite SNR information for positioning. 1) Max_SatxC_No: Maximum signal-to-noise ratio; 2) Min_SatxC_No: Minimum signal-to-noise ratio; 3) Avg_SatxC_No: Average signal-to-noise ratio.
GNSS_SNR_TypeDef **GNSS_SNR_NotUsePos**	Satellite SNR information that is in the field of view, but not used for positioning.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	This function needs to be called after Device_Open Note: The acquired SNR and the number of satellites is valid values only after GNSS locking, which is 0 by default.
Example	
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)**	
Description	
GNSS signal-to-noise ratio (optional support required), real-time acquisition, will occupy the data channel for a short time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
GNSS_SatDate_TypeDef* **GNSS_SatDate**	Return GNSS signal-to-noise ratio information.
GNSS_SatDate_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_GetGNSS_SatDate() function.

	Note: The obtained signal-to-noise ratio and number of satellites are valid only after GNSS is locked; the default value is 0.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	
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 SWP Mode (main functions)

10.1 SWP_ProfileDeInit

int SWP_ProfileDeInit(void Device, SWP_Profile_TypeDef* UserProfile_O)**	
Function description	
The function initializes the configuration in SWP mode configuration parameter collection. SWP_Profile_TypeDef defines all parameters in SWP mode such as frequency, reference level, resolution bandwidth, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
SWP_Profile_TypeDef* UserProfile_O	Pointer to the default profile.
SWP_Profile_TypeDef	
double StartFreq_Hz	Start frequency in Hz.
double StopFreq_Hz	Stop frequency in Hz.
double CenterFreq_Hz	Center frequency in Hz.
double Span_Hz	Frequency span in Hz.
double RefLevel_dBm	Reference level in dBm.
double RBW_Hz	Resolution bandwidth in Hz.
double VBW_Hz	Video bandwidth in Hz.
double SweepTime	When the sweep time mode is set to Manual, this parameter represents and absolute time value; when set to *N, it fuctions as a sweep time multiplier.
SWP_FreqAssignment_TypeDef FreqAssignment	Specify the frequency assignment of the sweep: StartStop or CenterSpan. 1) StartStop: Specify sweep range by start and stop frequency; 2) CenterSpan: Specify sweep range by center frequency and span.
Window_TypeDef Window	Specify the window function used for FFT analysis: 1) FlatTop: good amplitude accuracy; 2) Blackman_ Nuttall: good frequency resolution; 3) LowSideLobe: low sidelobe window.
RBWMode_TypeDef RBWMode	Configure RBW Update Method: 1) RBW_Manual: Manually set the RBW value; 2) RBW_Auto: Automatically adjusts RBW based on SPAN; 3) RBW_OneThousandthSpan: Forces RBW = 0.01*SPAN; 4) RBW_OnePercentSpan: Forces RBW = 0.01 * SPAN.

VBWMode_TypeDef VBWMode	Configure VBW Update Method: 1) VBW_Manual: Manually specify the VBW value; 2) VBW_EqualToRBW: Enforces VBW = RBW; 3) VBW_TenpercentRBW: Enforces VBW = 0.1 * RBW; 4) VBW_OnePercentRBW: Enforces VBW = 0.01 * RBW; 5) VBW_TenTimesRBW: Enforces VBW = 10 * RBW, effectively bypassing the VBW filter.
SweepTimeMode_TypeDef SweepTimeMode	Sweep Time Mode Configuration: 1) SWTMode_minSWT: Performs sweep with the minimun sweep time; 2) SWTMode_minSWTx2: Performs sweep with approximately *2 the minimum sweep time; 3) SWTMode_minSWTx4: Performs sweep with approximately *4 the minimum sweep time; 4) SWTMode_minSWTx10: Performs sweep with approximately *10 the minimum sweep time; 5) SWTMode_minSWTx20: Performs sweep with approximately *20 the minimum sweep time; 6) SWTMode_minSWTx50: Performs sweep with approximately *50 the minimum sweep time; 7) SWTMode_minSWTxN: Performs sweep with approximately *N the minimum sweep time, where N = SweepTimeMultiple; 8) SWTMode_Manual: Performs sweep with the manually specified sweep time, where sweep time = SweepTime; 9) SWTMode_minSMPxN: Performs sampling ata each individual frequency point with a duration approximately N times the minimum sampling time, where N = SampleTimeMultiple.
Detector_TypeDef Detector	Detector Mode Configuration: 1) Detector_Sample: No inter-frame detection for each frequency point's power spectrum; 2) Detector_PosPeak: Performs frame detection on each frequency point's power spectrum; 3) Detector_Average: Performs frame detection on each frequency point's power spectrum, outputting one frame with averaging applied across frames; 4) Detector_NegPeak: Performs frame detection on each frequency point's power spectrum, outputting one frame with MinHold applied across frames; 5) Detector_MaxPower: Before FFT, each frequency point undergoes

	extended sampling, and the frame with the maximum power is selected for FFT (only available in SWP mode for capturing transient signals like pulses); 6) Detector_RawFrames: Each frequency point is sampled multiple times, undergoes multiple FFT analyses, and outputs power spectra frame by frame (only available in SWP mode); 7) Detector_RMS: Performs frame detection on each frequency point's power spectrum, outputting one frame with RMS applied across frames.
TraceFormat_TypeDef TraceFormat	Trace Format Configuration: 1) TraceFormat_Standard: Frequency points are uniformly spaced (equal intervals); 2) TraceFormat_PrecisFrq: Frequency points are accurately aligned (non-uniform spacing for precise frequency representation).
TraceDetectMode_TypeDef TraceDetectMode	Trace Detection Mode Configuration (Frequency Axis): 1) TraceDetectMode_Auto: Automatically selects the optimal trace detection mode; 2) TraceDetectMode_Manua: Uses the manually specified trace detection mode.
TraceDetector_TypeDef TraceDetector	Trace Detector Type Configuration: 1) TraceDetector_AutoSample: Automatic sample detection; 2) TraceDetector_Sample: Sample detection; 3) TraceDetector_PosPeak: Positive peak detection; 4) TraceDetector_NegPeak: Negative peak detection; 5) TraceDetector_RMS: Root means square detection; 6) TraceDetector_Bypass: No detection performed; 7) TraceDector_AutoPeak: Automatic peak detection mode.
uint32_t TracePoints	The total points of sweep trace. The system will return the nearest available points according to the RBW and sweeping range.
TracePointsStrategy_TypeDef TracePointsStrategy	The strategy for setting the number of trace points: 1) SweepSpeedPreferred: Priority is given to faster sweep rate and return trace point is as close as possible to target trace points; 2) PointsAccuracyPreferred: Priority is given to ensuring that the actual number of trace points is close to the target trace points; 3) BinSizeAssigned: The priority is to ensure that the trace is generated at the set frequency interval.
TraceAlign_TypeDef TraceAlign	The strategy for setting trace aligning:

	1) NativeAlign：Natural frequency alignment; 2) AlignToStart: Aligns to start frequency.
FFTExecutionStrategy_TypeDef **FFTExecutionStrategy**	Specify the strategy for FFT executing: 1) Auto: automatically choose whether to use the CPU or FPGA; 2) Auto_CPUPreferred: automatically choose whether to use the CPU or FPGA, CPU first; 3) Auto_FPGAPreferred: automatically choose whether to use the CPU or FPGA for FFT calculations, FPGA first; 4) CPUOnly_LowResOcc: Mandatory use of CPU computing, low resource usage, Maximum number of FFT points: 256K; 5) CPUOnly_MediumResOcc: Mandatory use of CPU computing, medium resource usage, Maximum number of FFT points: 1M; 6) CPUOnly_HighResOcc: Mandatory use of CPU computing, high resource usage, Maximum number of FFT points: 4M; 7) FPGAOnly: Mandatory use of FPGA computing.
RxPort_TypeDef RxPort	Setting the signal receiving port (only supported by M60, M80, N60, N45): 1) ExternalPort: The receiver accepts data from the external input port; 2) InternalPort: The receiver accepts RF signals from the internal auxiliary signal source.
SpurRejection_TypeDef SpurRejection	Specify the strategy for spurious rejection: 1) Bypass: no additional rejection is provided; 2) Standard: a standard rejection is provided; 3) Enhanced: an enhanced rejection is provided; 4) The higher the spurious suppression level, the slower the sweep rate.
ReferenceClockSource_TypeDef **ReferenceClockSource**	Specify the referenc clock: 1) ReferenceClockSource_Internal: the internal reference clock (10MHz as default) is used; 2) ReferenceClockSource_External: the external reference clock (10MHz as default) is used, and if the external reference clock can not be locked, it will automatically switch to the internal reference clock; 3) ReferenceClockSource_Internal_Premium: the internal high quality (DOCXO or OCXO) clock source is used; 4) ReferenceClockSource_External_Forced: the external reference clock (10MHz as default) is used and will not change to the internal source even if it can not be locked.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz. It allows user to adjust

	frequency accuracy manually.
uint8_t EnableReferenceClockOut	Specify the reference clock output enable (Compatibility limited to E90/E200/N400 platforms).
SWP_TriggerSource_TypeDef TriggerSource	Sweep trigger source for RF receivers: 1) InternalFreeRun: Internal trigger free run; 2) ExternalPerHop: External trigger, each trigger jumps one frequency point; 3) ExternalPerSweep: External trigger, each trigger refreshes a trace.
TriggerEdge_TypeDef TriggerEdge	Specify the trigger edge: 1) RisingEdge: rising edge is effective; 2) FallingEdge: falling edge is effective; 3) DoubleEdge: both rising edge and falling edge are effective.
TriggerOutMode_TypeDef TriggerOutMode	Set the trigger output mode: 1) None: trigger out is disabled; 2) PerHop: a tirgger pluse will be sent once a frequency hop is completed; 3) PerSweep: a tirgger pluse will be sent once a full trace sweep is completed; 4) PerProfile: swtich output with each configuration change.
TriggerOutPulsePolarity_TypeDef TriggerOutPulsePolarity	Specify the pulse polarity of the output trigger: 1) Positive: positive pulse is used; 2) Negative: negative pulse is used.
uint32_t PowerBalance	Set dynamic power consumption control in SWP mode. The typical range is 0-5000, increasing this value will reduce the power consumption of the device but also slow down the sweep speed.
GainStrategy_TypeDef GainStrategy	Specify the gain strategy of the receiver: 1) LowNoisePreferred: optimized for low noise; 2) HighLinearityPreferred: optimized for high linearity.
PreamplifierState_TypeDef Preamplifier	Set the state of the preamplifier: 1) AutoOn: the preamplifier will be automatically enabled according to the reference level and atten; 2) ForcedOff: the preamplifer will be kept off regardless the reference level and atten.
uint8_t AnalogIFBWGrade	Specify the grade of analog IF bandwidth.
uint8_t IFGainGrade	Specify the gain grade of the IF.
int8_t Atten	Set the attenuation of receiver in dB. If the atten is set to -1, it will be ignored and the receiver will set gain state according to the reference

	level only. For value larger than -1, it has a higher priority than the reference level.
SWP_TraceType_TypeDef TraceType	Specify the trace type: 1) ClearWrite: the trace will be updated with clear and write method; 2) MaxHold: the trace will be updated with max hold method; 3) MinHold: the trace will be updated with max hold method; 4) ClearWriteWithIQ: the trace will be updated with clear and write method. The corresponding IQ data of each spectrum frame will also be attached.
LOOptimization_TypeDef Looptimization	Specify LO optimization: 1) LOOpt_Auto: Automatic LO optimization; 2) LOOpt_Speed: High sweep speed optimization; 3) LOOpt_Spur: Low spurious optimization; 4) LOOpt_PhaseNoise: Low phase noise optimization.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the SWP_GetPartialSweep() function for related examples.

10.2 SWP_Configuration

int SWP_Configuration(void Device, const SWP_Profile_TypeDef* ProfileIn, SWP_Profile_TypeDef* ProfileOut, SWP_TraceInfo_TypeDef* TraceInfo)**	
Description	
Configure the spectrometer device to SWP working mode and related parameters Parameters such as frequency, reference level, resolution bandwidth and other parameters in SWP mode are uniformly encapsulated in the SWP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const SWP_Profile_TypeDef* SWP_ProfileIn	Configuration Profile Input. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_Profile_TypeDef* SWP_ProfileOut	Configuration Profile Output. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_TraceInfo_TypeDef* TraceInfo	The information about the trace under the current configuration.
SWP_TraceInfo_TypeDef	

int FullsweepTracePoints	The points count of a full trace.
int PartialsweepTracePoints	The points count of each trace segment which can be fetched by single SWP_GetPartialSweep calling.
int TotalHops	The total hops for a full trace.
uint32_t UserStartIndex	The index of the closest point to the SWPProfile.StartFreq_Hz in the trace (Freq_Hz[]).
uint32_t UserStopIndex	The array index corresponding to the user-specified StopFreq_Hz in the trace array, where at HopIndex = 0, Freq[UserStartIndex] is the frequency point closest to SWPProfile.StartFreq_Hz.
double TraceBinBW_Hz	The interval frequency between two points of the trace.
double StartFreq_Hz	The frequency of the first point in the trace.
double AnalysisBW_Hz	Analysis bandwidth of a single hop.
int TraceDetectRatio	The TraceDetectRatio indicates how many raw data points are converted to a single output point.
int DecimateFactor	The decimate factor under the current configuration.
float FrameTimeMultiple	The device's analysis time at a single frequency point is calculated as: Acutal Analysis Time = Default Analysis Time (system preset) * Frame Time Multiplier. 1) Increasing the frame time multiplier will extend the device's minimum sweep time; 2) The relationship is not strictly linear.
double FrameTime	The frame time is the totoal analysis time at a single frequency point.
double EstimateMinSweepTime	The EstimateMinSweepTime is estimated minimum time for completing a full sweep. It is mainly affected by Span, RBW, VBW, frame time.
DataFormat_TypeDef DataFormat	Time-domain data format.
uint64_t SamplePoints	Time-domain data sampling length.
uint32_t GainParameter	Gain-related parameters include Space (31~24 Bit), PreAmplifierState (23~16 Bit), StartRFBand (15~8 Bit), and StopRFBand (7~0 Bit).
DSPPlatform_Typedef **DSPPlatform**	DSP Processing Platform Configuration: 1) CPU_DSP: Computation performed on CPU; 2) FPGA_DSP: Computation performed on FPGA.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Msut be called after SWP_ProfileDelnit.
Example	Please refer to the SWP_GetPartialSweep() function for related examples.

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)**	
Description	
the sweep mode configuration (SWPMode) provides optimized instrument settings based on application requirements. All sweep-related parameters including frequency, reference level, and resolution bandwidth are encapsulated in the SWP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
SWPApplicaton_TypeDef Application	Recommended Device Configurations for SWP Mode by Application Objective: 1) SWPNoiseMeas: Displayed average noise level measurement; 2) SWPChannelPowerMeas: Channel power measurement; 3) SWPOBMeas: Occupied bandwidth measurement; 4) SWPACPMeas: Adjacent channel power ratio measurement; 5) SWPIM3Meas: IP3/IM3 measurement.
const SWP_Profile_TypeDef* SWP_ProfileIn	Configuration Profile Input.
SWP_Profile_TypeDef* SWP_ProfileOut	Configuration Profile Output.
SWP_TraceInfo_TypeDef* TraceInfo	Return Spectrum Trace Information.
SWP_Profile_TypeDef	Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_TraceInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the SWP_Configuration() function.
uint8_t ifDoConfig	Return Command Success/Failure Status.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	1) When the ifDoConfig value is 0, it needs to call SWP_Configuration before execution; 2) When the ifDoConfig value is 1, the function will internally call SWP_Configuration itself, so no additional call to SWP_Configuration is required.
Example (the value of ifDoConfig is set to 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;
SWP_Profile_TypeDef 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)**	
Description	
Obtain the spectral results obtained at each frequency hopping point in SWP mode, and return the frequency hopping point sequence number, frame sequence number and auxiliary information of the measurement data, so that the user can stitch the results of multiple scans into the entire spectrum curve and return the function status.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
double Freq_Hz[]	The frequency array of the measurement data. The array size is equal to the TraceInfo.PartialsweepTracePoints.
float PowerSpec_dBm[]	The power array of the measurement data. The array size is equal to the TraceInfo.PartialsweepTracePoints.
int* HopIndex	Indicate the index of current hop in the whole hopping sequence.
int* FrameIndex	Indicate the index of current frame in all the frames.
MeasAuxInfo_TypeDef* MeasAuxInfo	Auxiliary measurement information.
MeasAuxInfo_TypeDef	
uint32_t MaxIndex	The index of the maximum power value within the data packet.
float MaxPower_dBm	The maximum power in the packet.
int16_t Temperature	The temperature of the device. Degree Celsius = 0.01 * Temperature.
double SysTimeStamp	System timestamp in second provided by the insystem timer.
double AbsoluteTimeStamp	Absolute timestamp provided by the insystem GNSS.
float Latitude	The latitude provided by the insystem GNSS.

float Longitude	The longitude provided by the insystem GNSS.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Msut be called after SWP_Configuration.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn, SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.StartFreq_Hz = 9e3;
SWP_ProfileIn.StopFreq_Hz = 6.35e9;
SWP_ProfileIn.RBWMode = RBW_Manual;
SWP_ProfileIn.RBW_Hz = 200e3;
Status = SWP_Configuration(&Device, &SWP_ProfileIn, &SWP_ProfileOut, &TraceInfo);
vector<double> Frequency(TraceInfo.FullsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.FullsweepTracePoints);
int HopIndex = 0, FrameIndex = 0;
MeasAuxInfo_TypeDef MeasAuxInfo;
for (int i = 0; i < TraceInfo.TotalHops; i++) {
    Status = SWP_GetPartialSweep(&Device, Frequency.data() + i * TraceInfo.PartialsweepTracePoints,
PowerSpec_dBm.data() + i * TraceInfo.PartialsweepTracePoints, &HopIndex, &FrameIndex, &MeasAuxInfo);
}
Device_Close(&Device);
```

10.5 SWP_GetFullSweep

int SWP_GetFullSweep(void Device, double Freq_Hz[], float PowerSpec_dBm[], MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Obtain the results of an entire spectrum in SWP mode and auxiliary information about the measurement data, and return the function status at the same time.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.

double Freq_Hz[]	The frequency array of the measurement data. The array size is equal to the TraceInfo.FullsweepTracePoints.
float PowerSpec_dBm[]	The power array of the measurement data. The array size is equal to the TraceInfo.FullsweepTracePoints.
MeasAuxInfo_TypeDef* MeasAuxInfo	Retrieve measurement metadata. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after SWP_Configuration.

Example

```
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 SWP Mode (other functions)

11.1 SWP_GetPartialSweep_PM1

int SWP_GetPartialSweep_PM1(void Device, SWPTrace_TypeDef* PartialTrace)**	
Description	
The polymorphic form of the SWP_GetPartialSweep, adjusting the form of the returned data on the basis of the original function to strengthen the encapsulation of the data.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
SWPTrace_TypeDef* PartialTrace	Return the top-level structure that contains configuration, return information, and staging data.
SWPTrace_TypeDef	
double* Freq_Hz	The address of the Frequency data that is temporarily stored in the Device pointer.
float* PowerSpec_dBm	The address of the PowerSpec_dBm data that is temporarily stored in the Device pointer.
int HopIndex	The frequency hopping frequency index of the data is used to stitch the spectrum.
int FrameIndex	Frame index of data for applications such as quasi-positive peak detection.
void* AlternIQStream	Address of time domain data (interleaved IQ form).
float ScaletoV	The coefficient from the time domain data to the absolute value of voltage (V).
MeasAuxInfo_TypeDef MeasAuxInfo	Return auxiliary information about the measurement data, as detailed in the SWP_GetPartialSweep description.
MeasAuxInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
SWP_Profile_TypeDef SWP_Profile	Configuration information for the data. Refer to the detailed definition of the structure parameter used in the SWP_ProfileDeInit() function.
SWP_TraceInfo_TypeDef SWP_TraceInfo	Trace information for the data. Refer to the detailed definition of the structure parameter used in the SWP_Configuration() function.
DeviceInfo_TypeDef DeviceInfo	Device information for the data. Refer to the detailed definition of the structure parameter used in the

	Device_Open() function.
DeviceState_TypeDef DeviceState	The device state corresponding to the data (defined in htra_api or in the previous Device_QueryDeviceState function). Refer to the detailed definition of the structure parameter used in the Device_QueryDeviceState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after SWP_Configuration.

Example

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef ProfileIn;
SWP_Profile_TypeDef ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
Status = SWP_ProfileDeInit(&Device, &ProfileIn);
Status = SWP_Configuration(&Device, &ProfileIn, &ProfileOut, &TraceInfo);
SWPTrace_TypeDef PartialTrace;
vector<double> Frequency(TraceInfo.PartialsweepTracePoints);
vector<float> PowerSpec_dBm(TraceInfo.PartialsweepTracePoints);
PartialTrace.Freq_Hz = Frequency.data();
PartialTrace.PowerSpec_dBm = PowerSpec_dBm.data();
Status = SWP_GetPartialSweep_PM1(&Device, &PartialTrace);
Status = Device_Close(&Device);
```

11.2 SWP_ResetTraceHold

void SWP_ResetTraceHold(void Device)**	
Description	
TraceType is MaxHold or MinHold, the retained trace data is reset.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	None.
Calling constraints	This parameter only applies when TraceType is set to either MaxHold or MinHold.

Example

```
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 Phase Noise Measurement Mode

12.1 PNM_ProfileDeInit

int PNM_ProfileDeInit(void Device, PNM_Profile_TypeDef* PNM_Profile)**	
Description	
Default configuration parameters for phase noise measurement.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
PNM_Profile_TypeDef* PNM_Profile	Phase noise measurement configuration structure.
PNM_Profile_TypeDef	
double CenterFreq	Center frequency.
float Threshold	Carrier detection threshold.
double RBWRatio	RBW ratio (segment RBW / segment start frequency).
double StartOffsetFreq	Start frequency offset.
double StopOffsetFreq	Stop frequency offset.
uint32_t TraceAverage	Trace smoothing count.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

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);**	
Description	
Configure phase noise measurement.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
const PNM_Profile_TypeDef* PNM_Profile_in	Phase noise measurement configuration structure (input).
PNM_Profile_TypeDef* PNM_Profile_out	Phase noise measurement configuration structure (output).
PNM_Profile_TypeDef	Refer to the detailed definition of the structure parameter used in the

	PNM_ProfileDeInit() function.
PNM_MeasInfo_TypeDef* **PNM_MeasInfo**	Phase noise measurement information structure.
PNM_MeasInfo_TypeDef	
uint32_t Segments	Frequency segment count (decade-based).
uint32_t TracePoints	Trace points.
uint32_t PartialUpdateCounts	Number of trace refreshes (Get function calls) per analysis.
uint32_t FramesInSegment [PHASENOISE_MAXFREQSEGMENT]	Frames per frequency segment (total).
uint32_t FrameDetRatioOfSegment [PHASENOISE_MAXFREQSEGMENT]	Frame detection ratio per segment.
double StartFreqOfSegment [PHASENOISE_MAXFREQSEGMENT]	Start frequency of each segment.
double StopFreqOfSegment [PHASENOISE_MAXFREQSEGMENT]	Stop frequency of each segment.
double RBWOfSegment [PHASENOISE_MAXFREQSEGMENT]	RBW per segment.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.3 PNM_StartMeasure

int PNM_StartMeasure(void Device)**	
Description	
Start phase noise measurement.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.4 PNM_StopMeasure

int PNM_StopMeasure(void Device)**

Description	
Force-stop phase noise measurement.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after PNM_StartMeasure.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.5 PNM_GetPartialUpdatedFullTrace

int PNM_GetPatialUpdatedFullTrace(void Device, double* CarrierFreq, float* CarrierPower, double Freq[], float PhaseNoise[], uint32_t FrameUpdateCounts[], MeasAuxInfo_TypeDef* MeasAuxInfo, float* RefLevel)**	
Description	
Get the phase noise measurement trace.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
double* CarrierFreq	Carrier frequency.
float* CarrierPower	Carrier power.
double Freq[]	Trace frequency axis (in Hz).
float PhaseNoise[]	Trace power axis (in dBc/Hz).
uint32_t FrameUpdateCounts[]	Refresh counter (index 0 for the farthest segment, N for the nearest segment; elements represent the number of refreshed frames per segment).
MeasAuxInfo_TypeDef* MeasAuxInfo	Auxiliary measurement information structure.
MeasAuxInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
float* RefLevel	Get the active reference level for measurement.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after PNM_Conguration.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.6 PNM_AutoSearch

int PNM_AutoSearch(void Device, double* CarrierFreq, uint8_t* Found)**	
Description	
Adavanced function: Detect signals exceeding carrier threshold via panoramic scan.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
double* CarrierFreq	Carrier frequency.
uint8_t* Found	Carrier detection indicator.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.7 PNM_Preset_FrameDetRatio

int PNM_Preset_FrameDetRatio(void Device, PNM_MeasInfo_TypeDef* MeasInfo)**	
Description	
Advanced function: Reset frame detection ratio for all frequency segments.	
Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
PNM_MeasInfo_TypeDef* MeasInfo	After resetting the frame detection ratio, update the measurement information structure for phase noise measurement.
PNM_MeasInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the PNM_Configuration() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after PNM_Configuration.
Example	Please refer to the PNM_Set_FrameDetRatio() function for related examples.

12.8 PNM_Set_FrameDetRatio

int PNM_Set_FrameDetRatio(void Device, uint32_t FrameDetRatioOfSegment[], PNM_MeasInfo_TypeDef* MeasInfo)**
Description
Advanced function: Manually configure frame detection ratio per frequency segment.

Compatibility	0.55.58 and later.
Parameter description	
void Device**	Device handle.
uint32_t FrameDetRatioOfSegment[]	Frame detection ratio configuration array (array length = total segments; index 0 corresponds to the farthest segment, index N to the nearest segment).
PNM_MeasInfo_TypeDef* MeasInfo	Refresh the phase noise measurement info struct information structure upon frame detection threshold modification.
PNM_MeasInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the PNM_Configuration() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after PNM_Configuration
Example	

```
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);
// Full-band carrier search (advanced feature, optional): you may use the PNM_AutoSearch interface to perform p
anoramic scanning and locate signals exceeding the carrier detection threshold
// double PeakFreq = 1.0;
// uint8_t Found = 0;
// Status = PNM_AutoSearch(&Device, &PeakFreq, &Found);
// If an ideal carrier is found, the frequency can be configured to the device for analysis using the PNM_Configura
tion interface
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;
Status = PNM_Configuration(&Device, &PNM_ProfileIn, &PNM_ProfileOut, &PNM_MeasInfo);
```

```
// Manually set frame detection ratio (advanced feature, optional): you may use the PNM_Set_FrameDetRatio int
erface to manually adjust the frame detection ratio for each frequency segment
// PNM_MeasInfo.FrameDetRatioOfSegment[PNM_MeasInfo.Segments - 1] = 10;
// Status = PNM_Set_FrameDetRatio(&Device, PNM_MeasInfo.FrameDetRatioOfSegment, &PNM_MeasInfo);
// Reset frame detection ratio (advanced feature, optional): you may use the PNM_Preset_FrameDetRatio interfa
ce to reset the frame detection ratio of each segment to default
// 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);
for (int i = 0; i < PNM_MeasInfo.PartialUpdateCounts; i++) // Retrieve the trace result of a phase noise measurem
ent
{
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 << "Carrier not found" << endl; // No signal above the carrier detection threshold found near the specified fr
equency point
break;
}
case APIRETVAL_WARNING_MeasUpdate :
{
i = 0;
```

```
cout << "Measurement state updated" << endl; // PNM special return value: during phase noise measurement, D
UT status changed, causing automatic update of measurement status, requiring PartialUpdateCounts data to be r
eacquired
break;
}
default: cout << "Please refer to the programming guide for general error codes" << endl;
}
}
}
cout << "wait";
}
PNM_StopMeasure(&Device);
Device_Close(&Device);
```

13 IQS Mode (main functions)

13.1 IQS_ProfileDeInit

int IQS_ProfileDeInit(void Device, IQS_Profile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the IQS mode. IQS_Profile_TypeDef defines all parameters in IQS mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
IQS_Profile_TypeDef* UserProfile_O	Pointer to the IQS configuration structure, used as an input/output variable.
IQS_Profile_TypeDef	
double CenterFreq_Hz	Center frequency in Hz.
double RefLevel_dBm	Reference level in dBm.
uint32_t DecimateFactor	Decimate factor. Effective analysis bandwidth = raw analysis bandwidth /decimate factor.
RxPort_TypeDef RxPort	Specify the input port of the receiver: 1) ExternalPort: external RF port is linked to the receiver; 2) InternalPort: the output of bult-in signal generator will be connected to the receiver through internal path and the external RF port is isolated. (Compatibility limited to M60/M80/N60/N45 platforms).
uint32_t BusTimeout_ms	Set the data transfer timeout in ms. The funcitions for data fetching will return an error if it fails to fetch data within the ButTimeOut.
IQS_TriggerSource_TypeDef TriggerSource	Specify the trigger source in the IQS mode: 1) External: Triggered by a physical signal connected to a trigger input port outside the device. 2) Bus: Triggered by means of a function (instruction); 3) Level: The device detects the input signal according to the set level threshold, and triggers automatically when the input exceeds the threshold; 4) Timer: Use the device internal timer to automatically trigger the set time period; 5) TxSweep: Data acquisition triggered by scanning from the device's internal signal source (ASG option required) When this trigger source is selected, the acquisition process will be triggered by the output trigger

	signal of the signal source scanning; 6) MultiDevSyncByExt: When the external trigger signal arrives, multiple machines perform synchronous trigger behavior; 7) MultiDevSyncByGNSS1PPS: On the arrival of GNSS-1PPS, the multicomputer does a synchronized trigger behavior; 8) GNSS1PPS: triggered by the 1PPS of the insystem GNSS(GNSS module option required).
TriggerEdge_TypeDef TriggerEdge	Specify the trigger edge: 1) RisingEdge: rising edge is effective; 2) FallingEdge: falling edge is effective; 3) DoubleEdge: both rising edge and falling edge are effective.
TriggerMode_TypeDef TriggerMode	Specify the trigger mode: 1) FixedPoint: data with a length specified by the TriggerLength will be acquired once the device is triggered; 2) Adaptive: once the device is triggered, the device will keep acquiring data until a stop signal is received. The stop signal may be an external trigger edge or a calling of the function IQS_BusTriggerStop.
uint64_t TriggerLength	When TriggerMode is set to FixedPoints, the trigger length specifies the total points of the data to be acquired for each trigger.
double TriggerLevel_dBm	When the TriggerSource is set to Level. The TriggerLevel specifies the threshold level of the trigger in dBm.
double TriggerLevel_SafeTime	When the TriggerSource is set to Level. The TriggerLevel_SafeTime specifies the saft time for a trigger in second. If the trigger signal can not maintain the active level for this time, the trigger will not be executed.
double TriggerDelay	Once triggered, the trigger action will be executed after this delay in second.
double PreTriggerTime	Pre-trigger time, s. Specify the pre-trigger time in second. Once triggered, pre-triggered data with a length of PreTriggerTime will be attached to the trigger data.
TriggerTimerSync_TypeDef TriggerTimerSync	Set the synchronization of the trigger timer: 1) NoneSync: no synchronization; 2) SyncToExt_RisingEdge: the timer will be continuously synchronized by the rising edge of external trigger; 3) SyncToExt_FallingEdge: the timer will be continuously synchronized by the falling edge of external trigger; 4) SyncToExt_SingleRisingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and

	will be synchronized for a once by the rising edge of external trigger; 5) SyncToExt_SingleFallingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the falling edge of external trigger; 6) SyncToGNSS1PPS_RisingEdge: the timer will be continuously synchronized by the rising edge of the 1PPS from the insystem GNSS; 7) SyncToGNSS1PPS_FallingEdge: the timer will be continuously synchronized by the falling edge of the 1PPS from the insystem GNSS; 8) SyncToGNSS1PPS_SingleRisingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the rising edge of the 1PPS from the insystem GNSS; 9) SyncToGNSS1PPS_SingleFallingEdge: after a calling of functioin IQS_SyncTimer_Single, the timer will be ready to be synchronizated and will be synchronized for a once by the falling edge of the 1PPS from the insystem GNSS.
double TriggerTimer_Period	Specify the period of timer trigger in second. It is effective when the TriggerSource is set to Timer.
uint8_t EnableReTrigger	Enable or disable the retrigger action: Once enabled, system will generate subsequence triggers automatically after the initial trigger. Retrigger is only available when TriggerMode is set to FixedPoint.
double ReTrigger_Period	Specify the retrigger period in second.
uint16_t ReTrigger_Count	Specify the counts of the retrigger after initial trigger.
DataFormat_TypeDef DataFormat	Output data format for IQ data： 1) Complex8bit: complex data in 8-bit format; 2) Complex16bit: complex data in 16-bit format; 3) Complex32bit: complex data in 32-bit format; 4) Real16bit: Real, single-channel data 16-bit; 5) Real32bit: Real, single-channel data-32bit; 6) Real8bit: Real, single-channel data-8bit.
GainStrategy_TypeDef GainStrategy	Specify the gain strategy of the receiver: 1) LowNoisePreferred: optimized for low noise; 2) HighLinearityPreferred: optimized for high linearity.
PreamplifierState_TypeDef Preamplifier	Set the state of the preamplifier: 1) AutoOn: the preamplifier will be automatically enabled according to the reference level and atten;

	2) ForcedOff: the preamplifer will be kept off regardless the reference level and atten.
uint8_t AnalogIFBWGrade	Specify the grade of analog IF bandwidth.
uint8_t IFGainGrade	Specify the gain grade of the IF. Larger grade leads to a higher IF gain.
ReferenceClockSource_TypeDef ReferenceClockSource	Specify the referenc clock： 1) ReferenceClockSource_Internal: the internal reference clock (10MHz as default) is used; 2) ReferenceClockSource_External: the external reference clock (10MHz as default) is used, and if the external reference clock can not be locked, it will automatically switch to the internal reference clock; 3) ReferenceClockSource_Internal_Premium: the internal high quality (DOCXO or OCXO） clock source is used; 4) ReferenceClockSource_External_Forced: the external reference clock (10MHz as default) is used and will not change to the internal source even if it can not be locked.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz.
uint8_t EnableReferenceClockOut	Specify the reference clock output enable (Compatibility limited to E90/E200/N400 platforms).
double NativeIQSampleRate_SPS	For devices with variable sampling rate capability, user can specify the native IQ sampling rate of the device.
int8_t Atten	Set the attenuation of receiver in dB. If the atten is set to -1, it will be ignored and the receiver will set gain state according to the reference level only. For value larger than -1, it has a higher priority than the reference level.
DCCancelerMode_TypeDef DCCancelerMode	Set the operating mode of DC canceler: 1) DCCOff: the DC canceler if off; 2) DCCHighPassFilterMode: canceler is on and high-pass filter method is applied, which has a good rejection of DC offset but also rejection for signal from DC to 100kHz; 3) DCCManualOffsetMode: canceler is on and manual bias method is applied, for which manual calibration is required but has no damge to the signal in DC; 4) DCCAutoOffsetMode: canceler is on and auto bias method is applied, for which manual calibration is not needed and has no damge to the signal in DC. If the device has a DC component at the centre frequency, please enable

	this function to suppress the DC component, if there is no DC component, there is no need to set this parameter.
QDCMode_TypeDef **QDCMode**	Set the operating mode of the QDC (quadrature demodulation corrector): 1) QDCOff: the QDC is off; 2) QDCManualMode: QDC is on and manual calibrate is needed. In this mode, the QDCIGain, QDCQGain, and QDCPhaseComp are configured based on user-defined manual settings; 3) QDCAutoMode: QDC is on and auto coefficients are used. In this mode, the QDCIGain, QDCQGain, and QDCPhaseComp are automatically configured to default values. If the amplitude-phase characteristics of the IQ data remain unchanged after enabling the QDC function, then this device does not require QDC activation.
float QDCIGain	Set the normalized linear gain of I channel. The setting range is 0.8~1.2 and 1.0 stands for no gain. QDCIGain is only effective when the QDCMode is set to QDCManualMode.
float QDCQGain	Set the normalized linear gain of Q channel. The setting range is 0.8~1.2 and 1.0 stands for no gain. QDCQGain is only effective when the QDCMode is set to QDCManualMode.
float QDCPhaseComp	Set the phase compensation coefficient of the QDC. The setting range is -0.2~+0.2. QDCPhaseComp is only effective when the QDCMode is set to QDCManualMode.
int8_t DCCIOffset	Set the DC bias of I channel. DCCIOffset is only effective when the DCCancelerMode is set to DCCManualOffsetMode.
int8_t DCCQOffset	Set the DC bias of Q channel. DCCIOffset is only effective when the DCCancelerMode is set to DCCManualOffsetMode.
LOOptimization_TypeDef **LOOptimization**	Set LO Optimization: 1) LOOpt_Auto: Auto LO optimization; 2) LOOpt_Speed: High-speed LO optimization; 3) LOOpt_Spur: LO-spur LO optimization; 4) LOOpt_PhaseNoise: LO-phase-noise LO optimization.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the IQS_GetIQStream() function for related examples.

13.2 IQS_Configuration

int IQS_Configuration(void Device, const IQS_Profile_TypeDef* ProfileIn, IQS_Profile_TypeDef***

ProfileOut, IQS_StreamInfo_TypeDef* StreamInfo)	
Description	
Set device to the IQS mode and configurate it with parameters specified in the IQS profile. In IQS mode, the LO signal remains fixed, and the system receives RF signals with a certain bandwidth centered at the LO frequency. Additionally, when the decimation factor is greater than 2, continuous time-domain recording can be achieved (requires a PC with specific configurations or higher).	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const IQS_Profile_TypeDef* IQS_ProfileIn	Configuration profile for IQS mode.
IQS_Profile_TypeDef* IQS_ProfileOut	Feedback profile from the system. Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
IQS_StreamInfo_TypeDef* StreamInfo	The information about the data stream under the current configuration.
IQS_StreamInfo_TypeDef	
double Bandwidth	Bandwith of the receiver's physical channel or digital signal processing under current configutation.
double IQSampleRate	IQ single-channel sampling rate under current configuration (unit: S/s).
uint64_t PacketCount	Total packet count under current configuration (effective only in FixedPoints mode).
uint64_t StreamSamples	In FixedPoints mode: represents the total sampling points under current configuration. In Adaptive mode: physically meaningless (value fixed to 0).
uint64_t StreamDataSize	In FixedPoints mode : indicates the total bytes to sample under current configuration. In Adaptive mode: no physical meaning (value fixed to 0).
uint32_t PacketSamples	Number of sampling points per data packet retrieved by IQS_GetIQStream.
uint32_t PacketDataSize	Number of meaningful data bytes returned by each IQS_GetIQStream() invocation.
uint32_t GainParameter	Gain-related parameters, including: 1) Space (Bits 31-24); 2) PreAmplifierState (Bits 23-16); 3) StartRFBand (Bits 15-8); 4) StopRFBand (Bits 7-0).

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_ProfileDelnit.
Example	Please refer to the IQS_GetIQStream() function for related examples.

13.3 IQS_BusTriggerStart

int IQS_BusTriggerStart(void Device)**	
Description	
Lanuch a bus trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_GetIQStream.
Example	Please refer to the IQS_GetIQStream() function for related examples.

13.4 IQS_BusTriggerStop

int IQS_BusTriggerStop(void Device)**	
Description	
End a bus trigger. When TriggerMode is configured as FixedPoints, the bus trigger operation initiated by IQS_BusTriggerStart() will automatically terminate upon reaching the specified trigger length without requiring an explicit call to the function for termination.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_GetIQStream.
Example	Please refer to the IQS_GetIQStream() function for related examples.

13.5 IQS_GetIQStream

int IQS_GetIQStream(void Device, void** AlternIQStream, float* ScaleToV, IQS_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
This function Return time-domain data, measurement parameters, and trigger details when operating in IQS mode.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
void AlternIQStream**	Memory address of time-domain data (interleaved IQ format). Each

	data packet has a fixed size of 64968 bytes. 1) When using int8_t IQ data type: I/Q channels each contain 32484 samples; each sample occupies 1byte; 2) When using int16_t IQ data type: I/Q channel each contain 16242 samples; each sample occupies 2 bytes; 3) When using int32_t IQ data type: I/Q channels each contain 8121 samples; each sample occupies 4 bytes; The IQ data is stored in interleaved format: IQIQ… .
float* ScaleToV	Coefficient for IQ data to the absolute voltage in volt.
MeasAuxInfo_TypeDef* **MeasAuxInfo**	Auxiliary measurement information. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
IQS_TriggerInfo_TypeDef* **TriggerInfo**	Trigger information.
IQS_TriggerInfo_TypeDef	
uint64_t **SysTimerCountOfFirstDataPoint**	The corresponding system timer count value for the first data point in the packet.
uint16_t **InPacketTriggeredDataSize**	The size of triggered data in the packet.
uint16_t **InPacketTriggerEdges**	The count of trigger edges in the packet.
uint32_t **StartDataIndexOfTriggerEdges[25]**	StartIndexs for the trigger edges.
uint64_t **SysTimerCountOfEdges[25]**	The corresponding system timer count value for every trigger edges.
int8_t EdgeType[25]	The polarity of trigger edges.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration.
Example	
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 IQS Mode (other functions)

14.1 IQS_MultiDevice_WaitExternalSync

int IQS_MultiDevice_WaitExternalSync(void Device, const IQS_Profile_TypeDef* ProfileIn)**	
Description	
Call this function and wait for more synchronous trigger signals.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const IQS_Profile_TypeDef* ProfileIn	IQS configures the structure pointer as an input variable.
IQS_Profile_TypeDef	Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration and requires TriggerSource to be set to either MultiDevSyncByExt or MultiDevSyncByGNSS1PPS.
Example	Please refer to the IQS_MultiDevice_Run() function for related examples.

14.2 IQS_MultiDevice_Run

int IQS_MultiDevice_Run(void Device)**	
Description	
Call this function to enable simultaneous operation of multiple devices.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_MultiDevice_WaitExternalSync.
Example	
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)**	
Description	
Call this function to initiate a timer-outer trigger single synchronization.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration and requires TriggerSource to be set to Timer.
Example	

```
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)**	
Description	
Obtain the time domain data in IQS mode, and the time domain data format is int8_t, int16_t, and int32_t, and you can select the data format according to your needs.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
IQStream_TypeDef* IQStream	IQ data stream, including IQ data and related configuration information.
IQStream_TypeDef	
void* AlternIQStream	Memory address of time-domain data (interleaved IQ format). Each data packet has a fixed size of 64968 bytes. 1) When using int8_t IQ data type: I/Q channels each contain 32484 samples; each sample occupies 1byte; 2) When using int16_t IQ data type: I/Q channel each contain 16242 samples; each sample occupies 2 bytes; 3) When using int32_t IQ data type: I/Q channels each contain 8121 samples; each sample occupies 4 bytes; The IQ data is stored in interleaved format: IQIQ…
float IQS_ScaleToV	The coefficient from the time domain data to the absolute value of voltage (V).
float MaxPower_dBm	The maximum power in the data.
uint32_t MaxIndex	The index of the maximum power in the data.
IQS_Profile_TypeDef IQS_Profile	Configuration information for the data.
IQS_Profile_TypeDef	Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
IQS_StreamInfo_TypeDef StreamInfo	Format information for the data.
IQS_StreamInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the IQS_Configuration() function.
IQS_TriggerInfo_TypeDef TriggerInfo	Trigger information for the data.

IQS_TriggerInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
DeviceInfo_TypeDef DeviceInfo	Device information for the data.
DeviceInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_Open()function.
DeviceState_TypeDef DeviceState	Device status information for data.
DeviceState_TypeDef	Refer to the detailed definition of the structure parameter used in the Device_QueryDeviceState() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration.
Example	
int Status = -1; int DeviceNum = 0; void* Device = NULL; **BootProfile_TypeDef BootProfile;** **BootProfile.DevicePowerSupply = USBPortAndPowerPort;** **BootProfile.PhysicalInterface = USB;** **BootInfo_TypeDef BootInfo;** **Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);** **IQS_Profile_TypeDef ProfileIn;** **IQS_Profile_TypeDef ProfileOut;** **IQS_StreamInfo_TypeDef StreamInfo;** **Status = IQS_ProfileDeInit(&Device, &ProfileIn);** **Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);** **IQStream_TypeDef IQStream;** **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)**	
Description	
Retrieve time-domain data and auxiliary measurement information in IQS mode, with selectable data formats (int8_t/ int16_t/ int32_t) per user requirements.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.

IQStream_TypeDef* IQStream	IQ data stream, including IQ data and related configuration information.
IQStream_TypeDef	Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream_PM1() function.
MeasAuxInfo_TypeDef* MeasAuxInfo	Return auxiliary measurement data information.
MeasAuxInfo_TypeDef	Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration.

Example

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn;
IQS_Profile_TypeDef ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
IQStream_TypeDef IQStream;
MeasAuxInfo_TypeDef MeasAuxInfo;
Status = IQS_BusTriggerStart(&Device);
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[])**	
Description	
Call this function to get the IQ time domain data with a data type of int16_t.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
int16_t IQ_data[]	An array of IQ data that receives 16 bits of single data.

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after IQS_Configuration.
Example	

```
int Status = -1; int DeviceNum = 0; void* Device = NULL;
BootProfile_TypeDef BootProfile;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
BootInfo_TypeDef BootInfo;
Status = Device_Open(&Device, DeviceNum, &BootProfile, &BootInfo);
IQS_Profile_TypeDef ProfileIn, ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &ProfileIn);
Status = IQS_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = IQS_BusTriggerStart(&Device);
vector<int16_t> IQ_Data(StreamInfo.StreamSamples);
Status = IQS_GetIQStream_Data(&Device, IQ_Data.data());
Status = Device_Close(&Device);
```

15 DET Mode

DET is a detection analysis mode that performs power detection on signals within a certain bandwidth to help users observe the level of the signal.

15.1 DET_ProfileDeInit

int DET_ProfileDeInit(void Device, DET_Profile_TypeDef* UserProfile_O)**	
Description	
This function initializes the configuration profile of the DET mode. DET_Profile_TypeDef defines all parameters in DET mode such as center frequency, reference level, decimate factor, etc.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
DET_Profile_TypeDef* UserProfile_O	Pointer to the DET configuration structure (input/output parameter).
DET_Profile_TypeDef	
double CenterFreq_Hz	Refer to the detailed definition of the structure parameter used in the IQS_ProfileDeInit() function.
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	Set up the detector: 1) Detector_Sample: the power spectrum of each frequency point is processed without interframe detection; 2) Detector_PosPeak: frame detection is performed on the power spectrum between each frequency point, ultimately outputting one frame where MaxHold is applied across frames; 3) Detector_Average: frame detection is performed on the power spectrum between each frequency point, ultimately outputting one frame where averaging is applied across frames; 4) Detector_NegPeak: frame detection is performed on the power spectrum between each frequency point, ultimately outputting one frame where MinHold is applied across frames; 5) Detector_RMS: frame detection is performed on the power spectrum between each frequency point, ultimately outputting one frame where RMS calculation is applied across frames;
uint16_t DET_TraceDetectRatio	Set DET trace detection ratio.
GainStrategy_TypeDef GainStrategy	Please refer to the function with same name in the IQS_ProfileDeInit() section.
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	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after Device_Open.
Example	Please refer to the DET_GetPowerStream() function for related examples.

15.2 DET_Configuration

int DET_Configuration(void Device, const DET_Profile_TypeDef* ProfileIn, DET_Profile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo)**	
Description	
The center frequency, reference level, decimation factor, and other parameters in DET mode are encapsulated in the DET_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const DET_Profile_TypeDef* DET_ProfileIn	DET configuration structure pointer (input parameter). Refer to the detailed definition of the structure parameter used in the DET_ProfileDeInit() function.
DET_Profile_TypeDef* DET_ProfileOut	DET configuration structure pointer (output parameter).
DET_Profile_TypeDef	Refer to the detailed definition of the structure parameter used in the DET_ProfileDeInit() function.
DET_StreamInfo_TypeDef* StreamInfo	Relevant information of DET data in DET mode.
DET_StreamInfo_TypeDef	
uint64_t PacketCount	Packet count of the power stream.
uint64_t StreamSamples	The total samples in the power stream. It is valid when the TriggerMode= Fixedpoints. It is invalid when TriggerMode= Adaptive, and value is set as 0.
uint64_t StreamDataSize	The data size of the power stream. It is valid when the TriggerMode= Fixedpoints. It is invalid when TriggerMode= Adaptive, and value is set as 0.
uint32_t PacketSamples	The total samples in the packet. It's also the number of samples can

	be obtained by every calling of function DET_GetPowerStream.
uint32_t PacketDataSize	The data size of the packet. It's also the data size can be obtained by every calling of function DET_GetPowerStream.
double TimeResolution	The time interval between adjacent data points, s.
uint32_t GainParameter	Gain parameter information: 1) Bits 31-24: Gain range; 2) Bits 23-16: Preamplifier status; 3) Bits 15-0: Frequency band information.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_ProfileDeInit.
Example	Please refer to the DET_GetPowerStream() function for related examples.

15.3 DET_BusTriggerStart

int DET_BusTriggerStart(void Device)**	
Description	
Lanuch a bus trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_GetPowerStream.
Example	Please refer to the DET_GetPowerStream() function for related examples.

15.4 DET_BusTriggerStop

int DET_BusTriggerStop(void Device)**	
Description	
The current bus trigger will be terminated. When the trigger mode is set to FixedPoints, the bus trigger initiated by the DET_BusTriggerStart function will automatically stop after reaching the predefined trigger length, eliminating the need to call this function.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_GetPowerStream.
Example	Please refer to the DET_GetPowerStream() function for related

	examples.

15.5 DET_GetPowerStream

int DET_GetPowerStream(void Device, float NormalizedPowerStream[], float* ScaleToV, DET_TriggerInfo_TypeDef* TriggerInfo, MeasAuxInfo_TypeDef* MeasAuxInfo)**	
Description	
Retrieves DET mode detection data, returning the integer-to-absolute amplitude scale factor and trigger-related information, where NormalizedPowerStream represents the value of ($\sqrt{I^2+Q^2}$).	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
float NormalizedPowerStream[]	the normalized power level ($\sqrt{I^2+Q^2}$).
float* ScaleToV	Coefficient for normalized power level to the absolute voltage (V).
DET_TriggerInfo_TypeDef* TriggerInfo	Trigger information of DET data. Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
MeasAuxInfo_TypeDef* MeasAuxInfo	Auxiliary measurement information. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_Configuration.
Example	

```
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)**	
Description	
Synchronize the trigger timer by external trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DET_Configuration and requires TriggerSource to be set as Timer.
Example	

```
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;
Status = DET_Configuration(&Device, &ProfileIn, &ProfileOut, &StreamInfo);
Status = DET_SyncTimerByExtTrigger_Single(&Device);
Status = Device_Close(&Device);
```

16 ZeroSpan Mode

The ZeroSpan mode is used for zero span analysis of signals, providing real-time power measurement of the signal at a specific frequency, helping users accurately capture instantaneous signal changes.

16.1 ZSP_ProfileDeInit

int ZSP_ProfileDeInit(void Device, ZSP_Profile_TypeDef* UserProfile_O)**	
Description	
Initialize the configuration parameters related to the ZeroSpan mode. The center frequency, reference level, decimation factor, and other parameters in ZeroSpan mode are uniformly encapsulated in the ZSP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
ZSP_Profile_TypeDef* UserProfile_O	Pointer to the ZeroSpan configuration structure, serving as an input/output variable.
ZSP_Profile_TypeDef	
double CenterFreq_Hz	Refer to the detailed definition of the structure parameter used in the DET_ProfileDeInit() function.
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	Resolution Bandwidth.
double VBW_Hz	Video Bandwidth.

VBWMode_TypeDef VBWMode	VBW Update Mode.
RBWFilterType_TypeDef RBWFilterType	RBW Filter Type.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the function ZSP_Configuration() function related examples.

16.2 ZSP_Configuration

int ZSP_Configuration(void Device, const ZSP_Profile_TypeDef* ProfileIn, ZSP_Profile_TypeDef* ProfileOut, DET_StreamInfo_TypeDef* StreamInfo)**	
Description	
Configure the related parameters of the ZeroSpan mode. The center frequency, reference level, decimation factor, and other parameters in ZeroSpan mode are uniformly encapsulated in the ZSP_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const ZSP_Profile_TypeDef* ProfileIn	Pointer to the ZSP configuration structure, used as an input variable. Refer to the detailed definition of the structure parameter used in the ZSP_ProfileDeInit() function.
ZSP_Profile_TypeDef* ProfileOut	Pointer to the ZSP configuration structure, used as an output variable. Refer to the detailed definition of the structure parameter used in the ZSP_ProfileDeInit() function.
DET_StreamInfo_TypeDef* StreamInfo	Related information of ZSP data in ZSP mode. Refer to the detailed definition of the structure parameter used in the DET_Configuration() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after ZSP_ProfileDeInit.
Example	

```
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 RTA Mode

RTA is a real-time spectrum analysis mode that helps users observe frequency hopping or transient burst signals.

17.1 RTA_ProfileDeInit

int RTA_ProfileDeInit(void Device, RTA_Profile_TypeDef* UserProfile_O)**	
Description	
Initialize and configure the related parameters of RTA mode. The center frequency, reference level, decimation factor, and other parameters in the RTA mode are uniformly encapsulated in the RTA_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
RTA_Profile_TypeDef* UserProfile_O	RTA configuration structure pointer, used as input/output variable.
RTA_Profile_TypeDef	
double CenterFreq_Hz	Please refer to parameters with same name in the SWP_ProfileDeInit() section.
double RefLevel_dBm	
double RBW_Hz	
double VBW_Hz	
RBWMode_TypeDef RBW_Mode	
VBWMode_TypeDef VBW_Mode	
uint32_t DecimateFactor	Set decimation factor, which determines the bandwidth of RTA.
Window_TypeDef Window	Please refer to parameters with same name in the SWP_ProfileDeInit() section.
SweepTimeMode_TypeDef SweepTimeMode	
double SweepTime	
Detector_TypeDef Detector	
TraceDetectMode_TypeDef TraceDetectMode	
TraceDetector_TypeDef TraceDetector	
uint32_t TraceDetectRatio	Trace detection ratio. The trace detector outputs 1 spectrum data point for every TraceDetectRatio original spectrum data points in the original spectrum trace.
RxPort_TypeDef RxPort	Please refer to parameters with same name in the SWP_ProfileDeInit() section.
uint32_t BusTimeout_ms	
RTA_TriggerSource_TypeDef TriggerSource	

TriggerEdge_TypeDef TriggerEdge	
TriggerMode_TypeDef TriggerMode	
double TriggerAcqTime	Sets the sampling time after input trigger, effective only in FixedPoints mode.
TriggerOutMode_TypeDef TriggerOutMode	Please refer to parameters with same name in the IQS_ProfileDeInit() section.
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	
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

17.2 RTA_Configuration

int RTA_Configuration(void Device, const RTA_Profile_TypeDef* ProfileIn, RTA_Profile_TypeDef* ProfileOut, RTA_FrameInfo_TypeDef* FrameInfo)**	
Description	
Configure the related parameters of the RTA mode. Parameters such as center frequency, reference level, and decimation factor in RTA mode are uniformly encapsulated in the RTA_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
const RTA_Profile_TypeDef* RTA_ProfileIn	Pointer to the RTA configuration structure, used as an input variable. Refer to the detailed definition of the structure parameter used in the RTA_ProfileDeInit() function.
RTA_Profile_TypeDef* RTA_ProfileOut	Pointer to the RTA configuration structure, used as an output variable. Refer to the detailed definition of the structure parameter used in the RTA_ProfileDeInit() function.
RTA_FrameInfo_TypeDef* StreamInfo	Related information of RTA data in RTA mode.
RTA_FrameInfo_TypeDef	
double StartFrequency_Hz	Start frequency in Hz.
double StopFrequency_Hz	Stop frequency in Hz.
double POI	Minimum signal duration time with 100% probability of interception in s.
double TraceTimestampStep	The timestamp step of each trace in each packet of data. (The overall packet timestamp is SysTimerCountOfFirstDataPoint in TriggerInfo)
double TimeResolution	The sampling time of each time-domain data, which is also the resolution of the timestamp

double PacketAcqTime	The acquisition time corresponding to each data packet
uint32_t PacketCounts	Packet count of the specturm stream. A specturm stream is generated once the device is triggered.
uint32_t PacketFrames	The number of valid frames in each data packet
uint32_t FFTSize	The number of points in the FFT of each frame
uint32_t FrameWidth	The number of points after FFT frame capture, which is also the number of points per Trace in the data packet, and can be used as the number of X-axis points (width) in the probability density map.
uint32_t FrameHeight	The spectral amplitude range corresponding to the FFT frame, which can be used as the number of Y-axis points (height) in the probability density map.
uint32_t PacketSamplePoints	The number of sampling points corresponding to each data packet.
uint32_t PacketValidPoints	The number of valid frequency domain data points contained in each data packet.
uint32_t MaxDensityValue	The upper limit of the element value at a single site in the probability density bitmap.
uint32_t GainParameter	Gain-related parameters, including Space (31~24Bit), PreAmplifierState (23~16Bit), StartRFBand (15~8Bit), StopRFBand (7~0Bit).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_ProfileDeInit.
Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

17.3 RTA_BusTriggerStart

int RTA_BusTriggerStart(void Device)**	
Description	
Lauch a bus trigger.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called before RTA_GetRealTimeSpectrum.
Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

17.4 RTA_BusTriggerStop

int RTA_BusTriggerStop(void Device)**	
Description	
Terminate the current bus trigger. When TriggerMode is set to FixedPoints, the bus trigger will automatically terminate after being initiated by the RTA_BusTriggerStart function and reaching the specified trigger length, without needing to call this function again.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_GetRealTimeSpectrum.
Example	Please refer to the RTA_GetRealTimeSpectrum() function for related examples.

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);**	
Description	
Obtain the real-time spectrum in RTA mode (without probability density plot).	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
uint8_t SpectrumStream[]	Pointer to the spectrum stream. The spectrum stream is consisted of contiguous spectral frames. The spectrum is in normalized relative power with LSB equals to 0.75dB. The array size is equal to the RTA_FrameInfo.PacketValidPoints.
RTA_PlotInfo_TypeDef* RTA_PlotInfo	The plot information structure returned by RTA after acquisition.
RTA_TriggerInfo_TypeDef* TriggerInfo	Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
MeasAuxInfo_TypeDef* MeasAuxInfo	Return auxiliary information of the measurement data; see the description of SWP_GetPartialSweep for details. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
RTA_PlotInfo_TypeDef	

float ScaleTodBm **float OffsetTodBm**	The scale factor and offset value for converting spectrum frame amplitude to dBm. The absolute power of the spectrum frame amplitude equals SpectrumStream[] * ScaleTodBm + OffsetTodBm.
uint64_t SpectrumBitmapIndex	The current probability density image segment's index within the complete triggered data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_Configuration.

Example

```
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,&MeasAuxInf
o);
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)**	
Description	
Obtain the real-time spectrum in RTA mode.	
Compatibility	0.55.0 and later.
Parameter description	

void Device**	Device handle.
uint8_t SpectrumStream[]	Pointer to the spectrum stream. The spectrum stream is consisted of contiguous spectral frames. The spectrum is in normalized relative power with LSB equals to 0.75dB. The array size is equal to the RTA_FrameInfo.PacketValidPoints.
uint16_t SpectrumBitmap[]	Return the probability density map bitmap. The size of this array equals RTA_FrameInfo.FrameHeight * RTA_FrameInfo.FrameWidth obtained via the RTA_Configuration function.
RTA_PlotInfo_TypeDef* **RTA_PlotInfo**	Structure of plotting information returned after RTA acquisition. Refer to the detailed definition of the structure parameter used in the RTA_GetRealTimeSpectrum_Raw() function.
RTA_TriggerInfo_TypeDef* **TriggerInfo**	Trigger-related information of RTA data. Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream() function.
MeasAuxInfo_TypeDef* **MeasAuxInfo**	Return auxiliary information of the measurement data; see the description of SWP_GetPartialSweep for details. Refer to the detailed definition of the structure parameter used in the SWP_GetPartialSweep() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_Configuration.

Example

```
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)**	
Description	
Call this function to initiate a timer-external single trigger synchronization.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after RTA_Configuration, and the TriggerSource must be set to Timer.
Example	

```
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;
Status = RTA_Configuration(&Device, &ProfileIn, &ProfileOut, &FrameInfo);
Status = RTA_SyncTimerByExtTrigger_Single(&Device);
Status = Device_Close(&Device);
```

18 Digital Demod(Option)

Consisting of the modulated signal spectrum, demodulated constellation diagram, eye diagram, and demodulation parameters, it deeply analyzes the modulation quality of the signal, provides multiple error metrics, and effectively evaluates the integrity and reliability of the signal during transmission.

18.1 Demod_Check

int Demod_Check ()	
Description	
Verify whether the demodulation library exists.	
Compatibility	0.55.55 and later.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	None
Example	
int Status = -1; **Status = Demod_Check();**	

18.2 Demod_Open

int Demod_Open (void Device)**	
Description	
Enable the demodulation function, check for the existence of a license, and allocate the required memory.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the Demod_Execute () function related examples.

18.3 Demod_Close

int Demod_Close (void Device)**	
Description	
Close the demodulation function.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Demod_Open.
Example	Please refer to the Demod_Execute() function for related examples.

18.4 Demod_Reset

int Demod_Reset (void Device)**	
Description	
Reset the demodulation function. When IQ data is discontinuous, Demod_Reset must be called before each call to Demod_Execute. If IQ data is continuous, only Demod_Execute needs to be called continuously.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Demod_Open.
Example	Please refer to the Demod_Execute() function for related examples.

18.5 Demod_GetVersion

int Demod_GetVersion (void Device, char version[])**	
Description	
Get the demodulation API version.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
char version[]	Return the demodulation API version.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Demod_Open.
Example	Please refer to the Demod_Execute() function for related examples.

18.6 Demod_DeInit

void Demod_DeInit (Demod_Profile_TypeDef* DemodProfile)	
Description	
Initialize the demodulation configuration structure, assigning initial values to each parameter in the structure.	
Compatibility	0.55.55 and later.
Parameter description	
Demod_Profile_TypeDef* DemodProfile	Demodulation configuration structure.
Demod_Profile_TypeDef	
uint64_t SamplePoints	Number of Sample Points.

double SampleRate	Sampling Rate, Hz.
double SymbolRate	Symbol Rate, sym/s.
Demod_ModType_TypeDef ModType	Set the modulation type of the signal to be demodulated. Support FSK2 / FSK4 / GMSK / BPSK / QPSK / PSK8 / QAM16 / ASK2 / QAM64 / AM / FM / PM / CW / LowerSideband / UpperSideband / QAM128 / QAM256.
Demod_FilterType_TypeDef FilterType	Filter type, currently only supports RootRaisedCosine RootRaisedCosine: Root Raised Cosine filter; RaisedCosine: Raised Cosine filter; Gaussian: Gaussian filter; Rectangular: Rectangular filter.
double FilterAlpha	Filter roll-off factor (currently only supports Root Raised Cosine, 0.01 <= Alpha <= 0.99)
Return value	None.
Calling Constraints	Must be called after Demod_Open.
Example	Please refer to the Demod_Execute() function for related examples.

18.7 Demod_Configuration

int Demod_Configuration (void Device, const Demod_Profile_TypeDef* DemodProfileIn, Demod_Profile_TypeDef* DemodProfileOut)**	
Description	
Configure demodulation parameters.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
const Demod_Profile_TypeDef* DemodProfileIn	Input demodulation configuration structure. Refer to the detailed definition of the structure parameter used in the Demod_DeInit() function.
Demod_Profile_TypeDef* DemodProfileOut	Output demodulation configuration structure; if the input configuration is unreasonable, the output configuration structure will be overwritten with reasonable parameters. Refer to the detailed definition of the structure parameter used in the Demod_DeInit() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Demod_Open.
Example	Please refer to the Demod_Execute() function for related examples.

18.8 Demod_Execute

int Demod_Execute(void Device, const IQStream_TypeDef* IQStream, DemodInfo_TypeDef* DemodInfo)**	
Description	
Execute demodulation function.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
const IQStream_TypeDef* IQStream	IQ data stream, including IQ data and related configuration information. Refer to the detailed definition of the structure parameter used in the IQS_GetIQStream_PM1() function.
DemodInfo_TypeDef* DemodInfo	Demodulation information structure, including: eye diagram, constellation diagram, EVM, etc. Note: all pointer variables in the structure will point to the internal space of the function, and users do not need to manually allocate memory externally.
DemodInfo_TypeDef* DemodInfo	Output Demodulation Information Structure.
DemodInfo_TypeDef	
double* eDiagram	Starting Memory Address of Eye Diagram Data
uint32_t eDiagram_Len	Length of Eye Diagram Data
double* I_constellation	Starting memory address of I path constellation data
double* Q_constellation	Starting memory address of Q path constellation data
uint32_t constellation_Len	Length of constellation data
int32_t* bitStream	Starting memory address of bit stream data
uint32_t bitStream_Len	Length of bit stream data
int32_t* symbol	Start memory address of bitstream data
uint32_t symbol_Len	Length of bitstream data
double* EVM	Start memory address of EVM data, also for FSK Error and ASK Error
uint32_t EVM_Len	Length of EVM data
double EVM_RMS	Root mean square EVM
double EVM_MAX	Peak EVM
double* PhaseError	Starting memory address of PhaseError data, unit: degrees
uint32_t PhaseError_Len	Length of PhaseError data
double PhaseError_RMS	Root mean square PhaseError, unit: degrees
double PhaseError_MAX	Peak PhaseError, unit: degrees
double* MagError	Starting Memory Address of Amplitude Error Data

uint32_t MagError_Len	Amplitude Error Data Length
double MagError_RMS	Root Mean Square Amplitude Error
double MagError_MAX	Peak Amplitude Error
double FreqError	Frequency Error, the frequency error of the carrier relative to the center frequency, also known as CarrFreqOffset, unit Hz
double IQ_Offset	IQ Offset, unit dB, only for PSK and QAM
double SNR	Signal-to-noise ratio, unit dB, only for PSK and QAM
double GainImb	IQ gain imbalance, unit dB, only for PSK and QAM
double QuadError	IQ quadrature skew error, unit degrees, only for PSK and QAM
double FSK_Deviation	FSK frequency deviation, unit Hz
double CarrPower	Carrier power, unit dBm, only for ASK
double ASK_Depth	ASK Modulation Depth
double AM_Depth	AM Modulation Depth
double FM_Deviation	FM Modulation Deviation, unit Hz
double* Phase	PM Demodulation Data Starting Memory Address
uint32_t Phase_Len	PM Demodulation Data Length
double* Freq	FM demodulation data start memory address
uint32_t Freq_Len	FM demodulation data length
double* Amp	AM demodulation data start memory address
uint32_t Amp_Len	AM demodulation data length
double* SSB	SSB demodulation data start memory address, this parameter is used for both upper sideband and lower sideband
uint32_t SSB_Len	SSB Demodulation Data Length
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Demod_Open.

Example

```
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; // Note: The demodulation function can currently only input IQ data
```

```
 of type int16
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();
     // If the IQ data is not consecutive each time you do demodulation, you need to call Demod_Reset first, th
en 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)	
Description	
Configure demodulation parameters.	
Compatibility	0.55.55 and later.
Parameter description	
Demod_ModType_TypeDef ModType	Demodulation Type
Demod_SymbolMap_TypeDef SymbolMap[1024]	Represents a single symbol in the symbol mapping table. This symbol map is used to represent the relationship between symbols and coordinates in the modulation and demodulation process.
uint32_t* MapNum	The number of available symbols in the symbol mapping table.
Demod_SymbolMap_TypeDef	
float I	X-axis coordinate, representing the real part of the symbol in the co mplex plane.
float Q	Y-axis coordinate, representing the imaginary part of the symbol in t he complex plane.
Return value	None.
Calling Constraints	None.
Example	
int Status = -1; **Demod_ModType_TypeDef ModType = QPSK;** **Demod_SymbolMap_TypeDef SymbolMap[1024];** **uint32_t MapNum = 0;** **Demod_GenSymbolMap(ModType, SymbolMap, &MapNum);**	

19 Pulse Det(Option)

19.1 Pulse_Open

int Pulse_Open (void Device)**	
Description	
Turn on the pulse detection function to detect whether a license exists and allocate the required memory.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the Pulse_Detect () function for related examples.

19.2 Pulse_Close

int Pulse_Close (void Device)**	
Description	
Disable the pulse detection function.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open and Pulse_Open.
Example	Please refer to the Pulse_Detect() function for related examples.

19.3 Pulse_Detect

int Pulse_Detect(void Device, const Pulse_Profile_TypeDef* Pulse_Profile, PulseInfo_TypeDef* PulseInfo)**	
Description	
Perform pulse detection on the acquired detection analysis data.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
const Pulse_Profile_TypeDef* Pulse_Profile	Input pulse detection data, including data to be detected, thresholds, etc.
PulseInfo_TypeDef* PulseInfo	Output pulse detection results, including pulse width, period, duty cycle, etc.

Pulse_Profile_TypeDef	
uint32_t ExpPulseNum	Expected number of pulses to acquire
Unit_TypeDef unit	Unit of pulse data: Voltage_V: V Power_dBm: dBm
float* Pulse	Starting memory address of pulse data, with data unit depending on unit
uint64_t PulseSize	Pulse Data Length
double TimeResolution_s	Pulse Data Time Resolution, unit seconds (s)
double DetThreshold	Pulse Detection Threshold, unit consistent with data
PulseInfo_TypeDef	
uint32_t ActPulseNum	Specify the dwell time in second for power or frequency sweep.
PulseTDParam_TypeDef* PulseTDParam	Pulse detection time-domain parameters start memory address.
PulseAMPParam_TypeDef* PulseAMPParam	Pulse amplitude parameters start memory address.
PulseEstParam_TypeDef* PulseEstParam	Pulse plotting parameters start address.
PulseStatsParam_TypeDef PulseStats	Pulse detection statistical parameters.
PulseTDParam_TypeDef	
double RiseTime	Rise Time.
double RiseEdge	Rising Edge.
double FallTime	Fall Time.
double FallEdge	Falling Edge.
double Width	Pulse Width.
double Period	Period.
float DutyCycle	Duty Cycle.
PulseAMPParam_TypeDef	
float TopLevel_dBm	Peak Level (dBm).
float TopLevel_V	Peak Level (V).
float BaseLevel_dBm	Base Level (dBm).
float BaseLevel_V	Based Level (V).
float TopToBaseRatio_dB	Top-to-Base Ratio (dB).
float TopToBaseDiff_V	Top-to-Base Difference (V).
float Droop_dB	Droop (dB).
float Droop_V	Droop (V).
float Overshoot_dB	Over Shoot (dB).

float Overshoot_V	Over Shoot (V).
float Ripple_dB	Ripple (dB).
float Ripple_V	Ripple (V).
PulseEstParam_TypeDef	
double Level_10pct_Index[2]	The array indices corresponding to the 10% level value, where 0 is the index of the rising edge and 1 is the index of the falling dege.
double Level_50pct_Index[2]	The array indices corresponding to the 50% level value, where 0 is the index of the rising edge and 1 is the index of the falling dege.
double Level_90pct_Index[2]	The array indices corresponding to the 90% level value, where 0 is the index of the rising edge and 1 is the index of the falling dege.
double Level_95pct_Index[2]	The array indices corresponding to the 95% level value, where 0 is the index of the rising edge and 1 is the index of the falling dege.
double Width_25pct_Index	The array indices corresponding to the 25% pulse width position.
double Width_75pct_Index	The array indices corresponding to the 75% pulse width position.
uint64_t Start_Index	Infferred starting array indices for signal and noise components.
uint64_t Size	Inferred data lengths for signal and nosie components.
float* Noise_dBm	Inferred starting memory address for noise data in dBm.
float* Noise_V	Inferred starting memory address for noise data in V.
float* Signal_dBm	Inferred starting memory address for signal data in dBm.
float* Signal_V	Inferred starting memory address for signal data in V.
PulseStatsParam_TypeDe	
double MinPRI	Minimum Period.
double MaxPRI	Maxmum Period.
double MeanPRI	Mean Period.
double MinPW	Minimum Pulse Width.
double MaxPW	Maxmum Pulse Width.
double MeanPW	Mean Pulse Width.
float PRIDeviationPercent	Period Deviation Percentage.
float PWDeviationPercent	Pulse Width Deviation Percentage.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open and Pulse_Open.
Example	
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,
   &ScaleToV, &TriggerInfo, &MeasAuxInfo);
}
vector<float> NormalizedPowerStream_dBm(NormalizedPowerStream.size());
for (int i = 0; i < StreamInfo.StreamSamples; i++)
{
   NormalizedPowerStream_dBm[i] = 10 * log10(20 * pow(NormalizedPowerStream[i] * ScaleToV, 2));
}
Status = Pulse_Open(&Device);
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)**	
Description	
Perform pulse detection function on the acquired IQ data.	
Compatibility	0.55.55 and later.
Parameter description	
void Device**	Device handle.
const Pulse_Profile_TypeDef* Pulse_Profile	Input pulse detection data, including data to be detected, thresholds, etc. Refer to the detailed definition of the structure parameter used in the Pulse_Detect() function.
PulseInfoPM1_TypeDef* PulseInfoPM1	Output pulse detection results, including pulse modulation type, pulse frequency, and phase information.
PulseInfoPM1_TypeDef	
uint32_t ActPulseNum	Actual Number of Acquired Pulses; according to this number, detection results of each pulse can be obtained from the pointer variable.
PulseTDParam_TypeDef* PulseTDParam	Starting Memory Address of Pulse Detection Time Domain Parameters. Refer to the detailed definition of the structure parameter used in the Pulse_Detect() function.
PulseAMPParam_TypeDef* PulseAMPParam	Starting memory address of pulse detection amplitude parameters. Refer to the detailed definition of the structure parameter used in the Pulse_Detect() function.
PulseEstParam_TypeDef* PulseEstParam	Starting memory address of pulse detection plotting parameters. Refer to the detailed definition of the structure parameter used in the Pulse_Detect() function.
PulseStatsParam_TypeDef PulseStats	Pulse detection statistical parameters. Refer to the detailed definition of the structure parameter used in the Pulse_Detect() function.
uint8_t* Mod	Pulse modulation type, 0 represents CW, 1 represents LFM.
PulseFreqPhaseParam_TypeDef* PulseFreqPhase	Frequency and phase information of the pulse
PulseFreqPhaseParam_TypeDef	
double FreqMean	Frequency mean value.
double FreqErrorRMS	Frequency Error.
double PhaseMean	Phase Mean.

double PhaseErrorRMS	Phase Error.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open and Pulse_Open.

Example

```
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 ASG (Option)

ASG is an auxiliary source function that can output monophonic signal or swept signal, this hardware option is only supported by N45/60, M60/80.

20.1 ASG_ProfileDeInit

int ASG_ProfileDeInit(void Device, ASG_Profile_TypeDef* Profile)**	
Description	
The function initializes the relevant parameters of the ASG function and initialize the analog signal source.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
ASG_Profile_TypeDef* Profile	Pointer to the default profile.
ASG_Profile_TypeDef	
double CenterFreq_Hz	Center frequency, unit Hz, effective when the signal source operates in SIG_Fixed mode; input range 1M-1GHz; step 1 Hz.
double Level_dBm	Output power, unit dBm, effective when the signal source operates in SIG_Fixed mode; input range -127 to -5 dBm; step 0.25 dB.
double StartFreq_Hz	Start frequency in frequency sweep mode, unit Hz, effective when the signal source operates in SIG_FreqSweep_* mode; input range 1M-1GHz; step 1Hz.
double StopFreq_Hz	The stop frequency in frequency sweep mode, unit: Hz, effective when the signal source operates in the SIG_FreqSweep_* mode; input range 1M-1GHz; step size 1Hz.
double StepFreq_Hz	Step frequency in frequency sweep mode, unit is Hz, effective when the signal source operates in SIG_FreqSweep_* mode; input range 1M-1GHz; step size 1Hz.
double StartLevel_dBm	Starting power in power sweep mode, unit is Hz.
double StopLevel_dBm	Stopping power in power sweep mode, unit is Hz.
double StepLevel_dBm	Step power in power sweep mode, unit is Hz.
double DwellTime_s	In frequency sweep mode or power sweep mode, unit is s. When the trigger mode is BUS, this is the scan dwell time in seconds, effective when the signal source operates in *Sweep* mode; input range 0-1000000; step size 1.
double ReferenceClockFrequency	Specify the reference clock frequency in Hz. It allows user to adjust frequency accuracy manually.
ReferenceClockSource_TypeDef ReferenceClockSource	Specify the referenc clock: 1) ReferenceClockSource_Internal: the internal reference clock (10MHz as

	default) is used; 2) ReferenceClockSource_External: the external reference clock (10MHz as default) is used, and if the external reference clock can not be locked, it will automatically switch to the internal reference clock; 3) ReferenceClockSource_Internal_Premium: the internal high quality (DOCXO or OCXO) clock source is used; 4) ReferenceClockSource_External_Forced: the external reference clock (10MHz as default) is used and will not change to the internal source even if it can not be locked.
ASG_Port_TypeDef Port	Specify the output port of generator: 1) ASG_Port_External: The signal is output to the external port 2) ASG_Port_Internal: The signal is output to the receiver through an internal path.
ASG_Mode_TypeDef Mode	Off, single frequency, frequency sweep (external trigger, synchronized to re ception), power sweep (external trigger, synchronized to reception): 1) ASG_Mute; 2) ASG_FixedPoint; 3) ASG_FrequencySweep; 4) ASG_PowerSweep.
ASG_TriggerSource_TypeDef TriggerSource	Specify the trigger source: 1) ASG_TriggerIn_FreeRun: free running; 2) ASG_TriggerIn_External: external trigger; 3) ASG_TriggerIn_Bus: timer triggering.
ASG_TriggerInMode_TypeDef TriggerInMode	Specify SG trigger in mode: 1) ASG_TriggerInMode_Null: free running; 2) ASG_TriggerInMode_SinglePoint: single point trigger (trigger a single frequency or power configuration); 3) ASG_TriggerInMode_SingleSweep: single sweep trigger (trigger a sweep that takes one cycle at a time); 4) ASG_TriggerInMode_Continous: continuous scan trigger (trigger a continuous signal).
ASG_TriggerOutMode_TypeDef TriggerOutMode	Specify SG trigger out mode: 1) ASG_TriggerOutMode_Null: trigger out is disabled; 2) ASG_TriggerOutMode_SinglePoint: a tirgger pluse will be sent once a frequency hop is completed; 3) ASG_TriggerOutMode_SingleSweep: a tirgger pluse will be sent once a full trace sweep is completed;

Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	Please refer to the ASG_Configuration() function for related examples.

20.2 ASG_Configuration

int ASG_Configuration(void Device, ASG_Profile_TypeDef* ProfileIn, ASG_Profile_TypeDef* ProfileOut,ASG_Info_TypeDef* ASG_Info)**	
Description	
Configure the device to ASGMode and related parameters such as center frequency, power, and dwell time in ASG mode are encapsulated uniformly in the ASG_Profile_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void Device**	Device handle.
ASG_Profile_TypeDef* ProfileIn	Pointer to the ASG configuration structure, used as an input variable. Refer to the detailed definition of the structure parameter used in the ASG_ProfileDeInit() function.
ASG_Profile_TypeDef* ProfileOut	Pointer to ASG configuration structure, used as an output variable. Refer to the detailed definition of the structure parameter used in the ASG_ProfileDeInit() function.
ASG_Info_TypeDef* ASG_Info	In ASG mode, ASG related information, used as an output variable.
ASG_Info_TypeDef	
uint32_t SweepPoints	Number of sweeping points.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling Constraints	Must be called after Device_Open.
Example	

```
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 Analog Modulation Demodulation (ADM)

ADM is the Analog Modulation Demodulation (ADM) interface. Users can call its functions to perform analog demodulation processing.

21.1 ADM_Open

void ADM_Open (void AnalogMod)**	
Description	
Enable the Analog function and allocate memory space for Analog-related data storage. This function must be called prior to invoking any other Analog functions. To operate multiple instances simultaneously, utilize additional Analog pointers for concurrent operations.	
Compatibility	0.55.63 and later.
Parameter description	
void ADM**	Pointer to the Analog memory space. This function retruns the memory address of the currently opened Analog function, which must be used as an index reference for subsequent API calls.
Return value	None.
Calling constraints	Must be called before any other Analog functions, and only once at the beginning; subsequent functions can then perform operations using the returned device memory address. For any non-exceptional ADM_Open call, ADM_Close must be called to release memory after the functionality is no longer needed.
Example	Please refer to the AMD_AMDemod_PM1() function related examples.

21.2 ADM_Close

void ADM_Close (void AnalogMod)**	
Description	
Disable the Analog function and release the allocated memory.	
Compatibility	0.55.63 and later.
Parameter description	
void ADM**	Pointer to the Analog memory space.
Return value	None.
Calling constraints	Call this function only at the end of program execution to shut down the Analog functionality and release memory. To reuse Analog features, invoke ADM_Open again to reinitialize the function.

Example	Please refer to the AMD_AMDemod_PM1() function for related examples.

21.3 ADM_AMDemod

int ADM_AMDemod(void ADM, const void* AlternIQStream, DataFormat_TypeDef DataFormat, uint64_t SamplePoints, double IQSampleRate, float DemodWaveform[])**	
Description	
Perform AM demodulation on the IQ data, returning only the demodulated waveform.	
Compatibility	0.55.0 and later.
Parameter description	
void ADM**	Memory space reference required to run Analog.
const void* AlternIQStream	Input IQ data. Please refer to the structure parameter definition with the same name in the IQS_GetIQStream function.
DataFormat_TypeDef DataFormat	Data format of IQ data. Please refer to the structure parameter definition with the same name in the IQS_ProfileDeInit function.
uint64_t SamplePoints	Length of the time-domain data.
double IQSampleRate	Sampling rate of a single IQ channel.
float DemodWaveform[]	AM-demodulated waveform data, with length equal to that of the IQ signal d ata.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after AMD_Open.
Example	Please refer to the AMD_AMDemod_PM1() function related examples.

21.4 ADM_FMDemod

int ADM_FMDemod(void ADM, const void* AlternIQStream, DataFormat_TypeDef DataFormat, uint64_t SamplePoints, double IQSampleRate, bool reset, float DemodWaveform[])**	
Description	
Perform AM demodulation on the IQ data.	
Compatibility	0.55.63 and later.
Parameter description	
void ADM**	Memory space reference required to run Analog.
const void* AlternIQStream	Input IQ data. Please refer to the structure parameter definition with the same name in the IQS_GetIQStream function.
DataFormat_TypeDef	Data format of IQ data.

DataFormat	Please refer to the structure parameter definition with the same name in the IQS_ProfileDeInit function.
uint64_t SamplePoints	Length of the time-domain data.
double IQSampleRate	IQ sampling rate.
float DemodWaveform[]	FM-demodulated waveform data, with length equal to that of the IQ signal data.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after AMD_Open.
Example	Please refer to the AMD_FMDemod_PM1() function related examples.

21.5 ADM_AMDemod_PM1

int ADM_AMDemod_PM1(void ADM, const void* AlternIQStream, DataFormat_TypeDef DataFormat, uint64_t SamplePoints, double IQSampleRate, float IQS_ScaleToV, AMDemodParam_TypeDef* AMDemodParam)**	
Description	
Perform AM demodulation on the IQ data, returning the demodulated waveform and modulation parameters.	
Compatibility	0.55.63 and later.
Parameter description	
void ADM**	Memory space reference required to run Analog.
const void* AlternIQStream	Input IQ data. Please refer to the structure parameter definition with the same name in the IQS_GetIQStream function.
DataFormat_TypeDef DataFormat	Data format of IQ data. Please refer to the structure parameter definition with the same name in the IQS_ProfileDeInit function.
uint64_t SamplePoints	Length of the time-domain data.
double IQSampleRate	IQ sampling rate.
float IQS_ScaleToV	Coefficient for converting time-domain data to voltage magnitude (V).
AM_DemodParam_TypeDef* AM_DemodParam	After FM demodulation, returns parameters such as the FM-modulated signal frequency and FM frequency deviation.
AM_DemodParam_TypeDef	
float* DemodWaveform	Demodulated waveform (%).
float* AFSpectrum_ModDepth	Magnitude of the audio spectrum, with length DemodWaveformSize / 2, using a linear scale, unit: %.
double* AFSpectrum_Freq	Frequency of the audio spectrum (Hz).
uint32_t DemodWaveformSize	Number of points in the demodulated waveform.
float ModDepth	Modulation depth (%).

float ModDepthPeakPos	Modulation depth of the positive peak, Peak+ (%).
float ModDepthPeakNeg	Average modulation depth of the negative peak, Peak- (%).
float ModDepthHalfPeak	Average modulation depth at half-peak, (Peak+ - Peak-) / 2 (%).
float ModDepthRMS	Average modulation depth based on RMS (%).
float CarrierPower	Carrier power (dBm).
double ModRate	Modulation rate (Hz).
float SINAD	Signal-to-noise and distortion ratio (SINAD) (dB).
float RMSPower	RMS power (dBm).
double FreqError	Frequency error (Hz).
float SNR	Signal-to-noise ratio (SNR) (dB).
float DistTotalVrms	Proportion of distortion voltage (RMS) (%).
float THD	Total harmonic distortion (THD) (%).
float PEP	Peak envelope power (PEP) (dBm).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after AMD_Open.

Example

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
Device_Open_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo);
IQStream_TypeDef IQStream
IQS_Profile_TypeDef IQS_ProfileIn;
IQS_Profile_TypeDef IQS_ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &IQS_ProfileIn);
IQS_ProfileIn.CenterFreq_Hz = 1e9;
IQS_ProfileIn.RefLevel_dBm = 0;
IQS_ProfileIn.DataFormat = Complex16bit;
IQS_ProfileIn.TriggerMode = FixedPoints;
IQS_ProfileIn.TriggerSource = Bus;
IQS_ProfileIn.DecimateFactor = 256;
IQS_ProfileIn.BusTimeout_ms = 5000;
IQS_ProfileIn.TriggerLength = 16242 * 1;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
```

```
IQS_Configuration_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
AMDemodParam_TypeDef AMDemodParam; //Demod
void* ADM = nullptr;
ADM_Open(&ADM);
vector<int16_t> IQ(StreamInfo.StreamSamples * 2);
vector<float> DemodWaveform(IQS_ProfileIn.TriggerLength);
while (1){
   uint32_t Points = StreamInfo.PacketSamples;
   Status = IQS_BusTriggerStart(&Device);
   for (uint32_t i = 0; i < StreamInfo.PacketCount; i++){
      Status = IQS_GetIQStream_PM1(&Device, &IQStream);
      if (Status) {
      std::cout << "IQS_GetIQStream_PM1: " << Status << std::endl;
      }
      else {
         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();
   ADM_AMDemod(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.DataFormat, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate, DemodWaveform.data());
   ADM_AMDemod_PM1(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.DataFormat, IQStream.IQS_Profile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate, IQStream.IQS_ScaleToV, &AMDemodParam);}
ADM_Close(&ADM);
Status = Device_Close(&Device);
```

21.6 ADM_FMDemod_PM1

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

Description
Perform FM demodulation on the IQ data, returning the demodulated waveform and modulation parameters.

Compatibility	0.55.63 and later.
Parameter description	
void ADM**	Memory space reference required to run Analog.
const void* AlternIQStream	Input IQ data. Please refer to the structure parameter definition with the same name in the IQS_GetIQStream function.
DataFormat_TypeDef DataFormat	Data format of IQ data. Please refer to the structure parameter definition with the same name in the IQS_ProfileDeInit function.
uint64_t SamplePoints	Length of the time-domain data.
double IQSampleRate	IQ sampling rate.
float IQS_ScaleToV	Coefficient for converting time-domain data to voltage magnitude (V).
bool reset	Reset the buffer. For a single-session demodulation, this should be set to 1 only on the first function call; subsequent calls should be set to 0.
FMDemodParam_TypeDef* FMDemodParam	After FM demodulation, returns parameters such as the FM-modulated signal frequency and FM frequency deviation.
FM_DemodParam_TypeDef	
float* DemodWaveform	Demodulated waveform (%).
float* AFSpectrum_ModDepth	Magnitude of the audio spectrum, with length DemodWaveformSize / 2, using a linear scale, unit: %.
double* AFSpectrum_Freq	Frequency of the audio spectrum (Hz).
uint32_t DemodWaveformSize	Number of points in the demodulated waveform.
float Deviation	Modulation frequency deviation (Hz).
float DeviationPeakPos	Modulation frequency deviation of the positive peak, Peak+ (Hz).
float DeviationPeakNeg	Average modulation frequency deviation of the negative peak, Peak- (Hz).
float DeviationHalfPeak	Average modulation frequency deviation at half-peak, (Peak+ - Peak-) / 2 (Hz).
float DeviationRMS	Average modulation frequency deviation based on RMS (Hz).
float CarrierPower	Carrier power (dBm).
double CarrierFreqErr	Carrier frequency deviation (Hz).
double ModRate	Modulation rate (Hz).
float SINAD	Signal-to-noise and distortion ratio (SINAD) (dB).
float SNR	Signal-to-noise ratio (SNR) (dB).
float DistTotalVrms	Ratio of distortion to total RMS (%).
float THD	Total harmonic distortion (THD) (%).
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after AMD_Open.

Example

```
int Status = 0;void* Device = NULL;int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
Device_Open_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo);
IQStream_TypeDef IQStream;
IQS_Profile_TypeDef IQS_ProfileIn;
IQS_Profile_TypeDef IQS_ProfileOut;
IQS_StreamInfo_TypeDef StreamInfo;
Status = IQS_ProfileDeInit(&Device, &IQS_ProfileIn);
IQS_ProfileIn.CenterFreq_Hz = 1e9
IQS_ProfileIn.RefLevel_dBm = 0;
IQS_ProfileIn.DataFormat = Complex16bit;
IQS_ProfileIn.TriggerMode = FixedPoints;
IQS_ProfileIn.TriggerSource = Bus;
IQS_ProfileIn.DecimateFactor = 256;
IQS_ProfileIn.BusTimeout_ms = 5000;
IQS_ProfileIn.TriggerLength = 16242 * 1;
Status = IQS_Configuration(&Device, &IQS_ProfileIn, &IQS_ProfileOut, &StreamInfo);
IQS_Configuration_ErrorHandling(Status, &Device, DevNum, &BootProfile, &BootInfo, &IQS_ProfileIn, &IQS_Pr
ofileOut, &StreamInfo);
FMDemodParam_TypeDef FMDemodParam; //Demod
void* ADM = nullptr;
ADM_Open(&ADM);
vector<int16_t> IQ(StreamInfo.StreamSamples * 2);
vector<float> DemodWaveform(IQS_ProfileIn.TriggerLength);
while (1){
   uint32_t Points = StreamInfo.PacketSamples;
   Status = IQS_BusTriggerStart(&Device);
   for (uint32_t i = 0; i < StreamInfo.PacketCount; i++){
      Status = IQS_GetIQStream_PM1(&Device, &IQStream);
      if (Status) {
         std::cout << "IQS_GetIQStream_PM1: " << Status << std::endl;
      }
      else {
```

```
        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();
   ADM_FMDemod(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.DataFormat, IQStream.IQS_Profil
e.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate, true,DemodWaveform.data());
   ADM_FMDemod_PM1(&ADM, IQStream.AlternIQStream, IQStream.IQS_Profile.DataFormat, IQStream.IQS_P
rofile.TriggerLength, IQStream.IQS_StreamInfo.IQSampleRate, IQStream.IQS_ScaleToV, true, &FMDemodPara
m);}
ADM_Close(&ADM);
Status = Device_Close(&Device);
```

22 Digital Signal Processing (Trace analysis)

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)	
Description	
This function analyzes the IM3 result of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP3_TypeDef* IM3Result	Return the measurement result of IP3.
TraceAnalysisResult_IP3_TypeDef	
double LowToneFreq	Frequency of the low tone in Hz
double HighToneFreq	Frequency of the high tone in Hz.
double LowIM3PFreq	Frequency of the low intermodulation product.
double HighIM3PFreq	Frequency of the high intermodulation product.
float LowTonePower_dBm	Power of the low tone in dBm.
float HighTonePower_dBm	Power of the high tone in dBm.
float TonePowerDiff_dB	Power difference between high tone and low tone in dB.
float LowIM3P_dBc	LowIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - LowTonePower_dBm, the strength of the low frequency intermodulation product relative to the strongest tone.
float HighIM3P_dBc	HighIM3P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - HighTonePower_dBm, the strength of the high frequency intermodulation product relative to the strongest tone.
float IP3_dBm	IP3 results in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	
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 Freq_Hz[], const float PowerSpec_dBm[], const uint32_t TracePoints, TraceAnalysisResult_IP2_TypeDef* IM2Result)	
Description	
This function analyzes the IM2 parameters of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
TraceAnalysisResult_IP2_TypeDef* IM2Result	Return the measurement result of IP2.
TraceAnalysisResult_IP2_TypeDef	
double LowToneFreq	Frequency of the low tone in Hz
double HighToneFreq	Frequency of the high tone in Hz.
double IM2PFreq	Frequency of the IM2 product in Hz.
float LowTonePower_dBm	Power of the low tone in dBm.
float HighTonePower_dBm	Power of the high tone in dBm.
float TonePowerDiff_dB	Power difference between the high tone and low tone in dB.
float IM2P_dBc	IM2P_dBc = max(LowTonePower_dBm, HighTonePower_dBm) - IM2P_dBm, the strength of the low freqeuncy product relative to the

	strongest tone.
float IP2_dBm	IP2 results in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.

Example

```
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);
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)

Description	
This function analyzes the channel power of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].

const double CenterFrequency	Specify the channel center frequency in Hz.
const double AnalysisSpan	Specify the channel bandwidth in Hz.
const double RBW	Specify the resolution bandwidth in Hz.
DSP_ChannelPowerInfo_TypeDef* ChannelPowerResult	Return the channel power measurement result.
DSP_ChannelPowerInfo_TypeDef	
float ChannelPower_dBm	Channel Power in dBm.
float PowerDensity	Power density in dBm/Hz.
float ChannelPeakIndex	The peak index within the channel.
double ChannelPeakFreq_Hz	Peak frequency within the channel in Hz.
float ChannelPeakPower_dBm	Peak power within the channel in dBm.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
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.FullsweepTracePoints, 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)	
Description	
This function analyzes the XdB bandwidth of the trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const float XdB	Specify X dB for analysis.
TraceAnalysisResult_XdB_TypeDef* XdBResult	Result for XdB anlaysis.
TraceAnalysisResult_XdB_TypeDef	
double XdBBandWidth_Hz	XdB bandwidth in Hz.
double CenterFreq_Hz	Center frequency in Hz of the signal.
double StartFreq_Hz	Start frequency in Hz of the signal.
double StopFreq_Hz	The stop frequency in Hz of the signal.
float StartPower_dBm	The power in dBm corresponding to the start frequency of the signal.
float StopPower_dBm	The power in dBm corresponding to the stop frequency of the signal.
uint32_t PeakIndex	Peak index within XdB bandwidth.
double PeakFreq_Hz	Peak frequency in Hz over XdB bandwidth.
float PeakPower_dBm	Peak power in dBm over XdB bandwidth.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	
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);
float XdB = 3;
TraceAnalysisResult_XdB_TypeDef XdBResult;
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)	
Description	
This function analyzes the occupied bandwidth of the trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const float OccupiedRatio	Specify the occupied bandwidth, usually set as 0.99.
TraceAnalysisResult_OBW_TypeDef* OBWResult	Results for occupation bandwidth analysis.
TraceAnalysisResult_OBW_TypeDef	
double OccupiedBandWidth	Bandwidth in Hz for given OccupiedRatio.
double CenterFreq_Hz	Center frequency in Hz of the signal.
double StartFreq_Hz	Start frequency in Hz of the signal.
double StopFreq_Hz	The stop frequency in Hz of the signal.
float StartPower_dBm	The power in dBm corresponding to the start frequency of the signal
float StopPower_dBm	The power in dBm corresponding to the stop frequency of the signal
float StartRatio	The proportion of power corresponding to the starting frequency of the occupied bandwidth.
float StopRatio	The proportion of power corresponding to the termination frequency that occupies the bandwidth.
uint32_t PeakIndex	Peak indexes within the consumed bandwidth

double PeakFreq_Hz	Peak frequency in Hz within the occupied bandwidth.
float PeakPower_dBm	Peak power in dBm within the occupied bandwidth.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	None.
Example	

```
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);
float OccupiedRatio = 0.99;
TraceAnalysisResult_OBW_TypeDef OBWResult;
Status = DSP_TraceAnalysis_OBW(Frequency.data(), PowerSpec_dBm.data(), TraceInfo.FullsweepTracePoints,
OccupiedRatio, &OBWResult);
Status = Device_Close(&Device);
```

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_TypeDef* ACPRResult)	
Description	
This function analyzers the adjacent channel power ratio (ACPR) of a trace.	
Compatibility	0.55.0 and later.
Parameter description	
const double Freq_Hz[]	Pointer to the frequency array.
const float PowerSpec_dBm[]	Pointer to the power array.
const uint32_t TracePoints	The size of Freq_Hz[] and PowerSpec_dBm[].
const DSP_ACPRFreqInfo_TypeDef	Specify the needed frequency information for ACPR analysis.

ACPRFreqInfo	
TraceAnalysisResult_ACPR_TypeDef* ACPRResult	Return the ACPR measurement result.
DSP_ACPRFreqInfo_TypeDef	
double RBW	Specify the resolution bandwidth in Hz for ACPR analysis.
double MainChCenterFreq_Hz	Specify the center frequency of main channel in Hz.
double MainChBW_Hz	Specify the bandwidth of main channel in Hz.
double AdjChSpace_Hz	Specify the channel space in Hz. Channel space is the frequency interval between the center frequency of the main channel and that of the adjacent channel.
uint32_t AdjChPair	1: one pair adjacent channels will be analyzed; 2: two pairs adjacent channels will be analyzed.
TraceAnalysisResult_ACPR_TypeDef	
float MainChPower_dBm	Power of the main channel in dBm.
uint32_t MainChPeakIndex	The peak index of the primary channel.
double MainChPeakFreq_Hz	The peak frequency in Hz of the primary channel.
float MainChPeakPower_dBm	Primary channel peak power in dBm.
double L_AdjChCenterFreq_Hz	Center frequency of the left adjacent channel in Hz.
double L_AdjChBW_Hz	Bandwidth of the left adjacency in Hz.
float L_AdjChPower_dBm	Power of the left adjacent channel in dBm.
float L_AdjChPowerRatio	Power ration of the left adjacent channel to the main channel.
float L_AdjChPowerDiff_dBc	Left adjacent channel power difference (left adjacent channel power - main channel power dBc).
float L_AdjChPeakIndex	The peak index of the left adjacent road.
double L_AdjChPeakFreq_Hz	The peak frequency in Hz of the left channel.
float L_AdjChPeakPower_dBm	Power of the left adjacent channel in dBm.
double R_AdjChCenterFreq_Hz	Center frequency of the right adjacent channel in Hz.
double R_AdjChBW_Hz	Bandwidth of the right adjacency in Hz.
float R_AdjChPower_dBm	Power of the right adjacent channel in dBm.
float R_AdjChPowerRatio	Power ration of the right adjacent channel to the main channel.
float R_AdjChPowerDiff_dBc	Right adjacent channel power difference (right adjacent channel power - main channel power dBc).
float R_AdjChPeakIndex	The peak index of the right adjacent road.
double R_AdjChPeakFreq_Hz	The peak frequency in Hz of the right channel.
float R_AdjChPeakPower_dBm	The value power in dBm of the right adjacent channel.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.

Calling constraints	None.
Example	
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, ACPRFreqInfo, 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)	
Description	
Performs emission template analysis on spectral data.	
Compatibility	0.55.62 and later.
Parameter description	
const DSP_SEMProfile_TypeDef*	SEM stencil reference line configuration.

semProfile	
const double Freq_Hz[]	An array of input frequencies.
const float PowerSpec_dBm[]	Input power array.
const uint32_t TracePoints	Enter the number of trace points.
const double CenterFreq	The center frequency of the input.
DSP_SEMResult_TypeDef* SEMResult	Returns measurements for all line segments of the SEM.
DSP_SEMProfile_TypeDef	
int mRefSetType	Reference Setting Type. mRefSetType = 0: peak reference; mRefSetType = 1: manual reference.
float mManualRefLevel	When mRefSetType = 1, the reference threshold is set manually.
DSP_SEMSegmentProfile_TypeDef mSegments[16]	Configure the start/cutoff frequency, start/cutoff threshold, etc. for each segment of the reference line, up to a maximum of 16 segments.
DSP_SEMSegmentProfile_TypeDef	
bool mState	Enable/disable SEM function Off = 0; Enable = 1.
double mStartFreq	Start frequency.
double mLimitStart	Start threshold.
double mStopFreq	Stop frequency.
double mLimitStop	Stop threshold.
int mMode	Absolute = 0; relative = 1.
int mPriority	Request = 0; Recommendation = 1.
DSP_SEMResult_TypeDef	
DSP_SEMSegmentResult_TypeDef mSegmentResults[16];	Returns the results of the SEM measurement.
DSP_SEMProfile_TypeDef mProfile	Returns the configuration at the time of the SEM measurement.
float mRefLevel	Returns the reference used for SEM measurements.
DSP_SEMSegmentResult_TypeDef	
double mLowerFreq	Low end frequencies.
float mLowerLevel	Low end level.
float mLowerMargin	Low-end residuals.
bool mLowerPassOrFail	Low end pass, fail. 0: pass; 1: fail.
double mUpperFreq	High-end frequency.
float mUpperLevel	High-end level.
float mUpperMargin	High-end residuals.
bool mUpperPassOrFail	High-end passes and fails.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.

Calling constraints	None.

Example

```
int Status = 0; void* Device = NULL; int DevNum = 0;
BootProfile_TypeDef BootProfile;
BootInfo_TypeDef BootInfo;
BootProfile.DevicePowerSupply = USBPortAndPowerPort;
BootProfile.PhysicalInterface = USB;
Status = Device_Open(&Device, DevNum, &BootProfile, &BootInfo);
SWP_Profile_TypeDef SWP_ProfileIn;
SWP_Profile_TypeDef SWP_ProfileOut;
SWP_TraceInfo_TypeDef TraceInfo;
SWP_ProfileDeInit(&Device, &SWP_ProfileIn);
SWP_ProfileIn.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){
      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);
      for (int i = 0; i < 3; i++) {
         std::cout << "low: " << result.mSegmentResults->mLowerPassOrFail << ", high: " << result.mSegmentRes
ults->mUpperPassOrFail << std::endl;
      }
```

```
    }
}
Device_Close(&Device);
```

23 Digital Signal Processing (processing of streaming)

23.1 DSP_Open

int DSP_Open(void DSP)**	
Description	
This function initializes inner variables and allocates memory space that is needed to excute DSP processing. This function must be called before the DSP handle to be used. Enable this function for DSP processing that requires different configurations.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space. After calling this function, it returns the memory address of the currently enabled DSP function. Subsequent API calls must use this reference to access the corresponding memory allocation.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	This function must be called once before calling any other DSP functions. Subsequent operations should use the returned device memory address. For every successful call to DSP_Open, you must call DSP_Close after completing all operations to release memory resources.
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.2 DSP_Close

int DSP_Close(void DSP)**	
Description	
This function turns off the DSP function and frees up memory space.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Invoke this function solely during program termination to disable DSP functionality and release memory resources; subsequent DSP operations require reinitialization via DSP_Open().
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.3 DSP_FFT_DeInit

int DSP_FFT_DeInit(DSP_FFT_TypeDef* IQToSpectrum)	
Description	
This function initializes the parameters for configuring FFT mode. The parameters including FFT points, window type, and decimate factor in FFT mode are uniformly encapsulated in the DSP_FFT_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
DSP_FFT_TypeDef* IQToSpectrum	Pointer to DSP_FFT configuration structure (input/output parameter).
DSP_FFT_TypeDef	
uint32_t FFTSize	FFT analysis points.
uint32_t SamplePts	The number of valid sampling points.
Window_TypeDef WindowType	the window functions employed in the FFT process: 1) FlatTop; 2) Blackman_Nuttall; 3) LowSideLobe: low sidelobe window
TraceDetector_TypeDef TraceDetector	Type of detector during video detection: 1) TraceDetector_AutoSample: Auto-sampling detection; 2) TraceDetector_Sample: Sample detection; 3) TraceDetector_PosPeak: Postive peak detecetion; 4) TraceDetector_NegPeak: Negative peak detection; 5) TraceDetector_RMS: Root-Mean-Square detection; 6) TraceDetector_Bypass: Bypass detection; 7) TraceDetector_AutoPeak: Automatic peak detection mode.
uint32_t DetectionRatio	Trace detection ratio.
float Intercept	Output spectrum interception. For example, Intercept = 0.8 to output 80% of the spectrum result.
bool Calibration	Whether calibration is performed.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called before DSP_FFT_Configuration.
Example	Please refer to the DSP_FFT_IQSToSpectrum() function for related examples.

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)**
Description

This function configures the parameters related to the FFT mode. The parameters such as FFT points, window type, and decimate factor in FFT mode are uniformly encapsulated in the DSP_FFT_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const DSP_FFT_TypeDef* ProfileIn	Pointer to DSP_FFT configuration structure (input parameter). Please refer to the function with same name in DSP_FFT_DeInit().
DSP_FFT_TypeDef* ProfileOut	Pointer to DSP_FFT configuration structure (output parameter). Please refer to the function with same name in DSP_FFT_DeInit().
uint32_t* TracePoints	The number of spectrum points that can be obtained under the current DSP_FFT configuration.
double* RBWRatio	Return the ratio of RBW to the sample rate. RBW = RBWRatio * IQSampleRate.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_FFT_DeInit.
Example	Please refer to the DSP_FFT_IQSToSpectrum() function for related examples.

23.5 DSP_FFT_IQSToSpectrum

int DSP_FFT_IQSToSpectrum(void DSP_FFT, const IQStream_TypeDef* IQStream, double Freq_Hz[], float PowerSpec_dBm[])**	
Description	
Convert IQ data into spectrum data, including frequency and power.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const IQStream_TypeDef* IQStream	Pointer to the default profile. Information about IQ streaming, including IQ data and related configuration information. Please refer to the function with same name in IQS_GetIQStream_PM1().
double Freq_Hz[]	The frequency array of the spectrum data. The array size is equal to TracePoints,which is output by the DSP_FFT_Configuration() function.
float PowerSpec_dBm[]	The power array for the spectrum data. The array size is equal to TracePoints,which is output by the DSP_FFT_Configuration() function.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_FFT_Configuration.

Example
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)	
Description	
This function initializes the parameters related to the DDC mode. The complex mixing and resampling parameters in DDC mode are packaged in a DSP_DDC_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
DSP_DDC_TypeDef* DDC_ProfileIn	Configuration profile for DSP_DDC mode.
DSP_DDC_TypeDef	
double DDCOffsetFrequency	The frequency offset value for the multimix.

double SampleRate	The sample rate of the multimix, it needs to be the same as the IQ data sampling rate.
float DecimateFactor	The re-sampling decimate factor, range: 1 ~ 2^{16}.
uint64_t SamplePoints	The number of sample points for multimixing.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called before DSP_DDC_Configuration.
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.7 DSP_DDC_Configuration

int DSP_DDC_Configuration(void DSP, const DSP_DDC_TypeDef* DDC_ProfileIn, DSP_DDC_TypeDef* DDC_ProfileOut)**	
Description	
This function configures the parameters related to the DDC mode. The complex mixing and resampling parameters in DDC mode are packaged in a DSP_DDC_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const DSP_DDC_TypeDef* ProfileIn	Pointer to DSP_DDC configuration structure (input parameter). Please refer to function with same name in DSP_DDC_DeInit() .
DSP_DDC_TypeDef* ProfileOut	Pointer to DSP_DDC configuration structure (output parameter). Please refer to function with same name in DSP_DDC_DeInit() .
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_DDC_DeInit.
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.8 DSP_DDC_Reset

void DSP_DDC_Reset(void DSP)**	
Description	
This function resets the cache in the DDC.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
Return value	None.
Calling constraints	Must be called after DSP_Open.
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.9 DSP_DDC_GetDelay

void DSP_DDC_GetDelay (void DSP, uint32_t* delay)**	
Description	
Retrieve DDC Group Delay.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
uint32_t* delay	Return the group delay of the decimation filter.
Return value	None.
Calling constraints	Must be called after DSP_DDC_Configuration.
Example	Please refer to the DSP_DDC_Execute() function for related examples.

23.10 DSP_DDC_Execute

int DSP_DDC_Excute(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**	
Description	
This function digitally downconverts IQ data.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const IQStream_TypeDef* IQStreamIn	Relevant information of the IQ streaming, including IQ data and related configuration information. Refer to the function with same name in IQS_GetIQStream_PM1().
IQStream_TypeDef* IQStreamOut	Output the relevant information of the IQ streaming, including IQ data and related configuration information. Refer to the function with same name in IQS_GetIQStream_PM1()
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_DDC_Configuration.
Example	

```
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);
// If the IQ data for each demodulation is not continuous, call DSP_DDC_Reset first, then call DSP_DDC_Execute.
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)	
Description	
This function analyzes the audio voltage (V), audio frequency (Hz), Sinnard (dB), and total harmonic distortion (%) parameters of the audio.	
Compatibility	0.55.0 and later.
Parameter description	
const double Audio[]	The array of audio signals.
const uint64_t SamplePoints	The length of the audio signal array.
const double SampleRate	The sample rate of the audio signal.
DSP_AudioAnalysis_TypeDef* AudioAnalysis	Return the measurement results for audio analysis.
DSP_AudioAnalysis_TypeDef	
double AudioVoltage	Audio voltage in V.
double AudioFrequency	Audio frequency in Hz.

double SINDA	Cinnard in dB.
double THD	Total harmonic distortion (%).
Return value	None.
Calling constraints	None.

Example

```
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)
Description

This function initializes the relevant parameters of LPF mode. The number of filter taps, cutoff frequency, stopband attenuation and other parameters in LPF mode are packaged in the Filter_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
Filter_TypeDef* LPF_ProfileIn	The pointer to the default profile.
Filter_TypeDef	
int n	Set the number of filter taps (n > 0).
float fc	Set the cutoff frequency (cutoff frequency/sample rate 0 < fc < 0.5).
float As	Set the stopband attenuation (As > 0, [dB]).
float mu	Set the fractional sampling offset (-0.5 < mu < 0.5).
uint32_t SamplePts	Set the number of Samples > 0.
Return value	None.
Calling constaints	Must be called before DSP_LPF_Configuration.
Example	Please refer to the DSP_LPF_Execute_Real() function for related examples.

23.13 DSP_LPF_Configuration

void DSP_LPF_Configuration(void DSP, const Filter_TypeDef* LPF_ProfileIn, Filter_TypeDef* LPF_ProfileOut)**	
Description	
This function configures the parameters related to the LPF mode. The number of filter taps, cutoff frequency, stopband attenuation and other parameters in LPF mode are packaged in the Filter_TypeDef structure.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const Filter_TypeDef* LPF_ProfileIn	Pointer to Filter_TypeDef configuration structure (input parameter). Refer to the function with same name in DSP_LPF_DeInit() .
Filter_TypeDef* LPF_ProfileOut	Pointer to Filter_TypeDef configuration structure (input parameter). Refer to the function with same name in DSP_LPF_DeInit()
Return value	None.
Calling constraints	Must be called after DSP_LPF_DeInit.
Example	Please refer to the DSP_LPF_Execute_Real() function for related examples.

23.14 DSP_LPF_Reset

void DSP_LPF_Reset(void DSP)**
Description

This function resets the cache in LPF.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_Open.
Example	Please refer to the DSP_LPF_Execute_Complex() function for related examples.

23.15 DSP_LPF_Execute_Real

void DSP_LPF_Execute_Real(void DSP, float Signal[], float LPF_Signal[])**	
Description	
This function performs low-pass filtering on the real signal.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
float Signal[]	Input signal.
float LPF_Signal[]	Low-pass filtered signal.
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_LPF_Configuration.
Example	

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

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

23.16 DSP_LPF_Execute_Complex

void DSP_LPF_Execute_Complex(void DSP, const IQStream_TypeDef* IQStreamIn, IQStream_TypeDef* IQStreamOut)**	
Description	
This function performs low-pass filtering on IQ signals.	
Compatibility	0.55.0 and later.
Parameter description	
void DSP**	The pointer to the DSP memory space.
const IQStream_TypeDef* IQStreamIn	Input IQ stream information, including IQ data and related configuration details. Refer to the function with same name in IQS_GetIQStream_PM1().
IQStream_TypeDef* IQStreamOut	Output IQ stream information, including IQ data and related configuration details. Refer to the function with same name in IQS_GetIQStream_PM1().
Return value	0: NoError. Nonzero: abnormal exsist, please refer to the Appendix 1.
Calling constraints	Must be called after DSP_LPF_Configuration.

Example

```
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 Appendix 1: API Return Value Index

Error Code	Cause of error/warning	Type[1]	Handling
0	No error	-	No processing is required, subsequent processes can be executed normally.
-1	Bus open error	Error	Check the device power supply, data line connection and check if the driver is installed correctly. After troubleshooting the error, you need to call Device_Open again to open the device.
-3	RF calibration file is missing[2]	Error	Check if the RF calibration file is placed in the specified directory. After troubleshooting the error, you need to call Device_Open again to open the device.
-4	IF calibration file is missing[2]	Error	Check if the IF calibration file is placed in the specified directory. After eliminating the error, you need to call Device_Open again to open the device.
-5	Device configuration information is missing[2]	Error	Check if the RF calibration file used is correct with the IF calibration file. After removing the error, you need to call Device_Open again to open the device.
-6	Device specification file is missing[2]	Error	Check that the device specification file (if required) is placed in the specified directory.
-7	Update Strategy failed	Error	Re-call Device_Open to open the device.
-8	Bus communication error	Error	Re-call the configuration function in the current mode.
-9	Data content error	Error	Re-call the configuration function in the current mode.
-10	Data not retrieved within specified time	Warning	Check whether the trigger source outputs the trigger signal normally, if there is no abnormality, continue to call the current function until the data is obtained.
-11	Configuration error via bus down	Warning.	Re-call the Configuration function to configure the device.
-12	Input signal amplitude exceeds the rated range in the current configuration	Warning	The current function gets to reduce the input signal amplitude or increase RefLevel_dBm as appropriate.
-14	The temperature has changed significantly since the last	Warning	The device temperature has changed significantly since the last configuration, it is recommended to re-call the Configuration function to configure the device for best

	configuration		performance.
-15	There is a locking exception in the local oscillator or clock	Warning	It is recommended to re-call the Configuration function to configure the device to try to restore the normal state.

[1] Type is "Error", you need to troubleshoot the problem immediately and turn the device back on, otherwise the device cannot continue to run subsequent processes. If the type is "warning", the device can continue the process without shutting down or reopening the device. However, it is still recommended to deal with the specific return value and the current application scenario selectively.

[2] For the return value of -3, -4, -5, or -6, you also need to confirm whether the file storage path is a full English path. If the path contains non-English characters, the API call will also indicate file loading failure.

www.harogic.com
info@harogic.com