V2.1.42
20/11/2026

H2 API Programming Guide

AROGIC

Contents

1. Version Information 7

2. Overview 8

3. Device and API version 9

4. Overview of Function Categories 10

5. API Calling Flow 11

5.1 CW Mode 11

5.2 Real-Time Streaming Mode 11

5.3 Playback Mode 11

6. Device and System Function 12

6.1 device_list_usb 12

6.2 device_ignore_pd 12

6.3 device_query_apiversion 13

6.4 device_open_usb 15

6.5 device_open_eth 16

6.6 device_close 17

6.7 device_preset 18

6.8 device_query_options 18

6.9 device_config_clock 19

6.10 device_query_clock 19

6.11 device_query_power_supply 20

6.12 device_config_func 21

6.13 device_config_fan 21

6.14 device_gpio_setbits 22

6.15 device_gpio_resetbits 22

6.16 device_config_power_state 23

6.17 device_query_power_state 23

6.18 device_query_temperature 24

6.19 device_config_gnss 24

6.20 device_query_gnss 25
6.21 device_query_gnss_data 25
6.22 device_config_trigger_out 26
6.23 device_query_discription_retval 28
7. Channel Setting Function 29
7.1 channel_preset 29
7.2 channel_start 29
7.3 channel_power_state 30
7.4 channel_config_lo_mode 30
7.5 channel_query_lo_mode 31
7.6 channel_config_trigger 31
7.7 channel_query_trigger 32
7.8 channel_bus_trigger 32
7.9 channel_config_port 33
7.10 channel_query_port 33
7.11 channel_tx_query_capabilities 34
7.12 channel_config_amplitude_compensation 34
8. Transmission Setting Functions 36
8.1 tx_config_output 36
8.2 tx_query_output 36
8.3 tx_config_ffm 37
8.4 tx_query_ffm 38
8.5 tx_config_fscan 39
8.6 tx_query_fscan 41
8.7 tx_config_lscan 41
8.8 tx_query_lscan 43
8.9 tx_config_mscan 44
8.10 tx_query_mscan 45
8.11 tx_clear_waveform 46

8.12 tx_download_waveform 46
8.13 tx_send_stream 51
8.14 tx_config_stream 53
8.15 tx_query_stream 53
9. Utility Functions 55
9.1 device_epoch_to_utc 55
10. Modulation Configuration Functions 56
10.1 mod_open 56
10.2 mod_close 56
10.3 mod_init_digitalmod 57
10.4 mod_init_am 57
10.5 mod_init_fm 58
10.6 mod_init_pm 58
10.7 mod_init_dsss 59
10.8 mod_init_pulse 59
10.9 mod_init_multitone 60
10.10 mod_init_stepsweep 60
10.11 mod_init_rampsweep 61
10.12 mod_init_ofdm 61
10.13 mod_init_awgn 62
10.14 mod_config_digitalmod 62
10.15 mod_config_am 65
10.16 mod_config_fm 65
10.17 mod_config_pm 66
10.18 mod_config_dsss 67
10.19 mod_config_pulse 67
10.20 mod_config_multitone 68
10.21 mod_config_stepsweep 69
10.22 mod_config_rampsweep 69

10.23 mod_config_ofdm 70
10.24 mod_config_awgn 71
10.25 mod_generate_stream 71
11. Structure Variables 73
11.1 eth_setting 73
11.2 gnss_setting 73
11.3 gnss_data 73
11.4 device_info 74
11.5 option 74
11.6 channel 74
11.7 stream_trigger 75
11.8 device_trigger_out 75
11.9 gain_setting 75
11.10 agc_setting 76
11.11 stream 76
11.12 data_organization 77
11.13 processer_det 77
11.14 processer_fft 77
11.15 aux_data 78
11.16 tx_stream 79
11.17 tx_channel_capabilities 79
11.18 digitalmod_profile 80
11.19 am_profile 81
11.20 fm_profile 81
11.21 pm_profile 81
11.22 dsss_profile 81
11.23 pulse_profile 82
11.24 multitone_profile 82
11.25 stepsweep_profile 83

11.26 rampsweep_profile 83
11.27 ofdm_profile 83
11.28 awgn_profile 83
12. Enumeration Variables 84
12.1 mod_digitalmod_type 84
12.2 mod_filter_type 84
12.3 mod_shape_type 85
13. Macro Definitions 86
13.1 Language 86
13.2 Option codes 86
13.3 Common state 86
13.4 Reference clock 86
13.5 GNSS constellation 86
13.6 Device interface type 87
13.7 Channel common definitions 87
13.8 Power definitions 87
13.9 RX gain and AGC 87
13.10 RX analyzer detectors 88
13.11 FFT window types 88
13.12 TX modes 88
13.13 Trigger Input Source 88
13.14 Trigger Output Source 88
13.15 Trigger Output Pulse Polarity 89
13.16 Trigger Channel Action 89
13.17 Digital Modulation Type 89
13.18 Filter Type 90
13.19 Modulation Wave 90
13.20 Multi-tone Phase Mode 90
Appendix 1: API Return Value Index 91

1. Version Information

Version Update Description Table

Version	Description	Date
V2.1.42	Added: SGF series software and firmware baseline versions.	20/11/2026
V2.0.40	Refactored: signal generator H2 API Interfaces. Added: Macro Definitions Section.	14/10/2026
V2.0.24	Added: device_gpio_setbits, device_gpio_resetbits.	16/07/2026
V2.0.18	Added: Modulation Configuration Functions section.	22/05/2026
V2.0.17	Add: device_query_state function.	30/04/2026
V2.0.12	Add: device_open_eth, device_config_fan, device_config_clock, device_query_clock, device_reset_clock_monitor, channel_config_trigger_out, channel_query_trigger_out, channel_config_lo_mode, channel_query_lo_mode, tx_config_lscan and tx_query_lscan function.	20/04/2026
V2.0.10	Add: device_list_usb, device_query_supply, device_config_gnss, device_query_gnss, device_query_gnss_info, device_epoch_to_utc, tx_config_fscan, tx_query_fscan and tx_config_stream function.	18/03/2026
V1.0	Initial version.	10/01/2026

2. Overview

This API system is a dynamic link library developed in C, used for programming and controlling the device. The signal generator mainly operates in the following three modes:

Continuous Wave Mode	In this mode, the device transmits a continuous wave, and no data needs to be sent from the user to the device.
Real-Time Streaming Mode	In this mode, the user can send IQ waveform data to the device via the USB interface. The device receives and plays the data in real time. When the user stops sending data, the device will finish playing all received data before stopping. The real-time sampling rate and data throughput are limited by the physical bandwidth of the data interface, while the total data size is not limited.
Playback Mode	This mode uses a pre-stored waveform playback mechanism. The user can first download IQ waveform data into the device memory, and then achieve multi-waveform playback by specifying the waveform index, number of playback loops, sampling rate, and the number of waveforms to be played. Since all waveform data is pre-stored, maximum analog bandwidth output can be achieved without relying on high-speed real-time data transmission. Note that the total amount of storable waveform data is limited by the device memory capacity.

3. Device and API version

The system operates through the coordinated execution of multiple software and firmware components. In this system, the involved components include: the main control firmware (MFW/MCU), FPGA firmware (FFW), bus firmware (Bus), and the application programming interface (API).

Version Matching Principle

To ensure stable system operation, the versions of the above components must strictly follow the baseline version alignment principle. Please refer to the table below for version verification and deployment.

Table 1 Software and Firmware Baseline Version Mapping Table (SGA/PGA Series)

API	FPGA	MCU	Bus
2.1.42	2.0.22	2.0.36	2.0.1
2.0.40	2.0.21	2.0.30	2.0.1
2.0.24	2.0.14	2.0.29	2.0.1
2.0.21	2.0.13	2.0.22	2.0.1
2.0.18	2.0.12	2.0.19	2.0.1
2.0.17	2.0.11	2.0.13	2.0.1

Table 2 Software and Firmware Baseline Version Mapping Table (SGF Series)

API	FPGA	MCU	Bus	Configuration
2.1.42	2.0.22	2.0.36	2.0.1	F60_R0 (std.)
	2.1.24	2.0.36	2.0.1	F60_R0 (opt.53)
	2.0.22	2.0.36	2.0.1	F60_R1 (std.)
	2.1.24	2.0.36	2.0.1	F60_R1 (opt.53)

Note:

1. To avoid system abnormalities, do not mix software and firmware from different baseline versions. Ensure that all components are deployed or upgraded strictly according to the versions listed in the table above.
2. Make sure that the API version in use matches the version indicated on the cover of the manual, to prevent inconsistencies between the documented interface descriptions and the actual functionality.

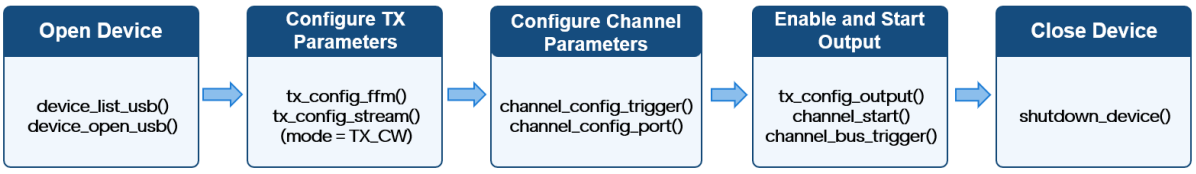
4. Overview of Function Categories

Table 3 API Function Categories

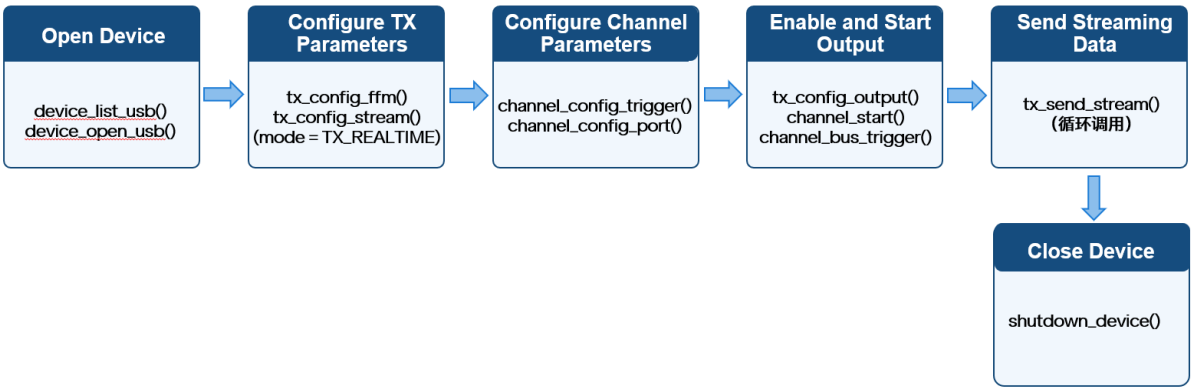
Function Category	Description
Device and System	Global functions that can be called in any operating mode. This category includes device open/close, global settings, retrieving device information, and obtaining device status.
Channel Settings	Channel-level control for managing the device's independent signal paths. Functions allow configuration, enabling, parameter setting, and status querying for specified channels, forming the basis for multi-channel synchronous or independent control.
Transmission Settings	Controls the transmission chain. Functions in this category configure transmission parameters, manage signal transmission processes, handle transmission data, and query transmission status. This is the core of the signal generator functionality.

5. API Calling Flow

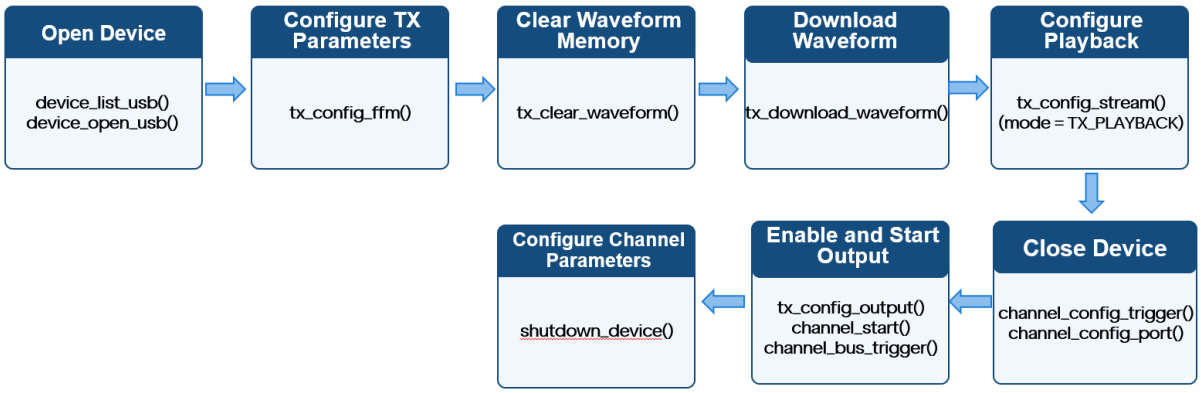
5.1 CW Mode



5.2 Real-Time Streaming Mode



5.3 Playback Mode



6. Device and System Function

device_open_usb must be called to initialize the device before accessing any hardware-related API functions. Upon completion of the application logic, call device_close to terminate the session and release allocated memory resources.

6.1 device_list_usb

int device_list_usb(device_info* device_info_o, uint32_t* count)	
Description	
Enumerate USB devices attached to the host.	
Compatibility	2.0.39 and later.
Parameter description	
[out] **device_info_o**	Return USB device information array. Refer to the device_info structure definition for details.
[out] **count**	Return total number of USB devices.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	None.
Example	Refer to the relevant example of the device_query_apiversion function.

6.2 device_ignore_pd

int device_ignore_pd (uint8_t dnum)	
Description	
Force the device to be powered via USB. It is used to bypass USB power capability negotiation and force USB power supply.	
Compatibility	2.0.39 and later.
Parameter description	
[in] **dnum**	USB device index.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	Must be called before device_open_usb initialization.
Example	Refer to the relevant example of the device_query_apiversion function.

6.3 device_query_apiversion

int device_query_apiversion (char version_name_o[16])	
Description	
Query API version.	
Compatibility	2.0.39 and later.
Parameter description	
[out] version_name_o	Return version string.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	None.
Example	

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
constexpr uint32_t max_devices = 8;
constexpr uint16_t max_options = 32;

char api_version[16]{};
status = device_query_apiversion(api_version);
std::printf("API version: %s\n", api_version);

device_info usb_devices[max_devices]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);
std::printf("USB device count: %u\n", device_count);

uint16_t chs = usb_devices[dnum].chn;
device_info dinfo{};
channel ch[MAXCHANNELS]{};
status = device_open_usb(dnum, chs, &device, ch, &dinfo);

for (uint8_t i = 0; i < dinfo.versions; ++i)
std::printf("%s\n", dinfo.version_name[i]);

option options[max_options]{};
uint16_t option_count = 0;
status = device_query_options(&device, options, &option_count);
for (uint16_t i = 0; i < option_count; ++i)
std::printf("Option %u: %s, state=%u\n", options[i].code, options[i].description,
options[i].state);
```

```
gnss_setting gnss{};
gnss.antenna = 0;
gnss.constellations = CONSTELLATION_GPS | CONSTELLATION_GLONASS |
CONSTELLATION_BEIDOU | CONSTELLATION_GALILEO;
gnss.state_pps_out = STATE_ON;
gnss.pps_x = 1.0f;
gnss.pps_delay = 0.0f;
status = device_config_gnss(&device, &gnss);
gnss_setting gnss_o{};
status = device_query_gnss(&device, &gnss_o);
gnss_data gnss_data_o{};
status = device_query_gnss_data(&device, &gnss_data_o);
status = device_config_power_state(&device, POWER_ON);

uint8_t power_state = POWER_OFF;
status = device_query_power_state(&device, &power_state);
status = device_config_fan(&device, 60.0f);
float voltage = 0.0f;
float current = 0.0f;
status = device_query_power_supply(&device, USB_RAIL, &voltage, &current);
std::printf("USB rail: %.3f V, %.3f A\n", voltage, current);

status = device_query_power_supply(&device, RF_RAIL, &voltage, &current);
std::printf("RF rail: %.3f V, %.3f A\n", voltage, current);
float temperature = 0.0f;
status = device_query_temperature(&device, 0, &temperature);
std::printf("Temperature: %.2f C\n", temperature);

status = device_config_clock(&device, REF_EXT, 10.0e6, REF_OUT_ON);

uint8_t state_ext_ref;
double fref;
uint8_t state_ref_out;
uint8_t is_locked;
status = device_query_clock(&device, &state_ext_ref, &fref, &state_ref_out, &is_locked);
if (status != H2_NOERROR) {
switch (status) {
case H2_WARNING_SYSCLK_UNLOCKED:
std::printf("Warning: system clock is unlocked.\n");
break;
case H2_WARNING_ADDACLK_UNLOCKED:
std::printf("Warning: AD/DA clock is unlocked.\n");
break;
case H2_WARNING_DLREF_UNLOCKED:
```

```
std::printf("Warning: LO reference clock is unlocked.\n");
break;
case H2_WARNING_RFLO_UNLOCKED:
std::printf("Warning: RF/IF LO is unlocked.\n");
break;
default:
std::printf("device_query_clock failed, status = %d.\n", status);
break;
}
}
return device_close(&device);
```

6.4 device_open_usb

```
int device_open_usb(
    uint8_t dnum,
    uint16_t chs,
    void** device_o,
    channel ch_o[],
    device_info* dinfo_o
)
```

Description	
Open a USB device and obtain channel information.	
Compatibility	2.0.39 and later.
Parameter description	
[in] dnum	USB device index.
[in] chs	Number of channels to enumerate
[out] device_o	Return the device handle.
[out] ch_o	Return channel array. Refer to the channel structure definition for details.
[out] dinfo_o	Return device information. Refer to the device_info structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	None.
Example	Refer to the relevant example of the device_query_apiversion function.

6.5 device_open_eth

int device_open_eth(**const eth_setting* settings,** **uint16_t chs,** **void** device_o,** **channel ch_o[],** **device_info* dinfo_o** **)**	
Description	
Open an ETH device and obtain channel information.	
Compatibility	2.0.39 and later.
Parameter description	
[in] settings	ETH device startup configuration. Refer to the eth_setting structure definition for details.
[in] chs	Number of channels to enumerate.
[out] device_o	Return device handle.
[out] ch_o	Return channel array. Refer to the channel structure definition for details.
[out] dinfo_o	Return device information. Refer to the device_info structure definition for details.
Return Value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	None.
Example	

```
int status;
void* device = nullptr;

constexpr int16_t strm = 0;
constexpr uint16_t chs = 1;
device_info dinfo{};
channel ch[MAXCHANNELS]{};

eth_setting eth{};
eth.is_ipv6 = 0;
eth.ip_address[0] = 192;
eth.ip_address[1] = 168;
eth.ip_address[2] = 1;
eth.ip_address[3] = 100;
eth.remote_port = 5000;
eth.read_timeout = 5000;
status = device_open_eth(&eth, chs, &device, ch, &dinfo);
```

```
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(&ch[0], fc, level);
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
setting.fc_offset = 0.0;
setting.phase_offset = 0.0f;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(&ch[0], strm, &setting, &trg);

status = channel_config_trigger(&ch[0], strm, TRIGGER_ACTION_HOP);

status = tx_config_output(&ch[0], STATE_ON, STATE_OFF);
status = channel_start(&ch[0]);

status = channel_bus_trigger(&ch[0], 1U << strm);
std::this_thread::sleep_for(std::chrono::seconds(1));

return device_close(&device);
```

6.6 device_close

int device_close(void device)**	
Description	
Close the device and release related resources.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function should only be invoked at the end of program execution. Upon invocation, the device connection is closed and allocated resources are released. To reuse the device, device_open_* must be called again to re-establish the connection and initialize the hardware.
Example	Refer to the relevant example of the device_list_usb function.
Example	Refer to the relevant example of the device_query_apiversion function.

6.7 device_preset

int device_preset(void device)**	
Description	
Device preset (including all-channel preset).	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

6.8 device_query_options

int device_query_options(**void** device,** **option option_o[],** **uint16_t* options_o** **)**	
Description	
Query option status.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[out] option_o	Option array. Refer to the option structure definition for details.
[out] options_o	Number of active options.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.9 device_config_clock

int device_config_clock(void** device, uint8_t state_ext_ref, double fref, uint8_t state_ref_out)	
Description	
Configure device reference clock.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] state_ext_ref	Reference clock source. Values: REF_INT (internal) / REF_EXT (external).
[in] fref	Reference clock frequency (Hz).
[in] state_ref_out	Reference clock output switch. Values: REF_OUT_OFF / REF_OUT_ON.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.10 device_query_clock

int device_query_clock(void** device, uint8_t* state_ext_ref_o, double* fref_o, uint8_t* state_ref_out_o, uint8_t* is_locked_o)	
Description	
Query device reference clock configuration.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[out] state_ext_ref_o	Current reference clock source. Values: REF_INT / REF_EXT.

[out] fref_o	Current reference clock frequency (Hz).
[out] state_ref_out_o	Reference clock output switch state. Values: REF_OUT_OFF / REF_OUT_ON.
[out] is_locked_o	Clock lock state: 0 unlocked, 1 locked.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.11 device_query_power_supply

```
int device_query_power_supply(
    void** device,
    uint8_t rail,
    float* voltage_o,
    float* current_o
)
```

Description	
Query voltage and current of the specified power rail (USB_RAIL / RF_RAIL).	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] rail	Power rail, values: USB_RAIL / RF_RAIL.
[out] voltage_o	Return voltage (V).
[out] current_o	Return current (A).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.12 device_config_func

int device_config_func (**void** device,** **uint8_t func,** **int8_t state** **)**	
Description	
Configure a generic device function.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] func	Function ID. Value: POWER_STATE.
[in] state	State. Values: POWER_ON / POWER_OFF / POWER_SAVING.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.13 device_config_fan

int device_config_fan(**void** device,** **float threshold** **)**	
Description	
Set fan configuration.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] threshold	Fan start temperature threshold: +INFINITY turns the fan off, -INFINITY keeps it always on, other values indicate fan start temperature.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.14 device_gpio_setbits

int device_gpio_setbits(**void** device,** **uint16_t bits** **)**	
Description	
Set GPIO bits.	
Compatibility	2.0.37 and later.
Parameter description	
[in] device	Device handle.
[in] bits	GPIO control bits, 16 bits in total; IO pins with the corresponding bit set to 1 are driven high.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

6.15 device_gpio_resetbits

int device_gpio_resetbits(**void** device,** **uint16_t bits** **)**	
Description	
Reset GPIO bits.	
Compatibility	2.0.37 and later.
Parameter description	
[in] device	Device handle.
[in] bits	GPIO control bits, 16 bits in total; IO pins with the corresponding bit set to 1 are driven low.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

6.16 device_config_power_state

int device_config_power_state(**void** device,** **uint8_t state** **)**	
Description	
Configure system power state.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] state	Power state. Values: POWER_ON / POWER_OFF / POWER_SAVING.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.17 device_query_power_state

int device_query_power_state(**void** device,** **uint8_t* state_o** **)**	
Description	
Query system power state.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[out] state_o	Return power state. Values: POWER_ON / POWER_OFF / POWER_SAVING.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.18 device_query_temperature

int device_query_temperature(void** device, uint8_t domain, float* temperature_o)	
Description	
Query current device temperature.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] domain	Temperature domain.
[out] temperature_o	Return device temperature (degrees Celsius).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.19 device_config_gnss

int device_config_gnss(void** device, const gnss_setting* gnss)	
Description	
Configure device GNSS parameters.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[in] gnss	GNSS configuration parameters. Refer to the gnss_setting structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.20 device_query_gnss

int device_query_gnss(void** device, gnss_setting* gnss_o)	
Description	
Query current device GNSS configuration.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[out] gnss_o	Return device GNSS configuration. Refer to the gnss_setting structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.21 device_query_gnss_data

int device_query_gnss_data(void** device, gnss_data* data_o)	
Description	
Query current device GNSS information.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
[out] data_o	Return device GNSS information. Refer to the gnss_data structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the device_query_apiversion function.

6.22 device_config_trigger_out

<table>
<tr><td colspan="2">int device_config_trigger_out(
void** device,
const device_trigger_out* trg
)</td></tr>
<tr><td colspan="2">Description</td></tr>
<tr><td colspan="2">Configure the device trigger output.
TX trigger output is configured on a per-device basis. When source is set to CHANNEL0 or CHANNEL1, recounter determines whether the trigger action is HOP or SWEEP. When source is set to TRIGGER_IN_ROUTE, the raw trigger event is output.</td></tr>
<tr><td>Compatibility</td><td>2.0.39 and later.</td></tr>
<tr><td colspan="2">Parameter description</td></tr>
<tr><td>[in] device</td><td>Device handle.</td></tr>
<tr><td>[in] trg</td><td>Trigger output configuration. Refer to the device_trigger_out structure definition for details.</td></tr>
<tr><td>Return value</td><td>0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.</td></tr>
<tr><td>Calling constraints</td><td>This function must be called after device_open_usb or device_open_eth.</td></tr>
<tr><td colspan="2">Example</td></tr>
<tr><td colspan="2">int status;
void* device0 = nullptr;
void* device1 = nullptr;

device_info dinfo0{};
device_info dinfo1{};
channel ch0[MAXCHANNELS]{};
channel ch1[MAXCHANNELS]{};
constexpr uint16_t chs = 1;
constexpr int16_t strm = 0;

eth_setting eth_settings[2]{};
for (auto& settings : eth_settings) {
settings.is_ipv6 = 0;
settings.ip_address[0] = 192;
settings.ip_address[1] = 168;
settings.ip_address[2] = 1;
settings.ip_address[3] = 100;
settings.read_timeout = 5000;
}</td></tr>
</table>

```
eth_settings[0].remote_port = 5000;
eth_settings[1].remote_port = 5001;

status = device_open_eth(&eth_settings[0], chs, &device0, ch0, &dinfo0);
status = device_open_eth(&eth_settings[1], chs, &device1, ch1, &dinfo1);

status = device_config_clock(&device0, REF_INT, 100.0e6, REF_OUT_ON);
status = device_config_clock(&device1, REF_EXT, 100.0e6, REF_OUT_OFF);

double fc0 = 3.1e9;
float level0 = 0.0f;
status = tx_config_ffm(ch0, fc0, level0);

double fc1 = 3.3e9;
float level1 = 0.0f;
status = tx_config_ffm(ch1, fc1, level1);

tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
stream_trigger trg0{};
trg0.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(ch0, strm, &setting, &trg0);

stream_trigger trg1{};
trg1.source = TRIGGER_SOURCE_EXTERNAL_RISING_EDGE;
status = tx_config_stream(ch1, strm, &setting, &trg1);

status = channel_config_trigger(ch0, strm, TRIGGER_ACTION_HOP);
status = channel_config_trigger(ch1, strm, TRIGGER_ACTION_HOP);

device_trigger_out trgout{};
trgout.state = STATE_ON;
trgout.source = TRIGGER_IN_ROUTE;
trgout.negative_pulse = TRIGGER_PULSE_POSITIVE;
status = device_config_trigger_out(&device0, &trgout);

status = tx_config_output(ch0, STATE_ON, STATE_OFF);
status = tx_config_output(ch1, STATE_ON, STATE_OFF);

status = channel_start(ch0);
status = channel_start(ch1);

std::this_thread::sleep_for(std::chrono::milliseconds(10));
```

```
status = channel_bus_trigger(ch0, 1U << strm);

std::this_thread::sleep_for(std::chrono::seconds(1));

status = device_close(&device0);
status = device_close(&device1);

return 0;
```

6.23 device_query_discription_retval

```
int device_query_discription_retval(
    int retval,
    int8_t language,
    char* discription,
    char* suggestion
)
```

Description	
Query return-value description and suggestion.	
Compatibility	2.0.39 and later.
Parameter description	
[in] retval	Function return value. Refer to the H2_ERROR_* / H2_WARNING_* and H2_WARNING_* definitions.
[in] language	Language parameter. Values: H2_LANGUAGE_ENGLISH / H2_LANGUAGE_CHINESE (reserved field, currently ignored and always outputs English).
[out] discription	Description string, no more than 128 bytes (including terminator).
[out] suggestion	Suggestion string, no more than 128 bytes (including terminator).
Return value	H2_NOERROR for success, H2_WARNING_PARAMOUTRANGE for invalid parameter.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

7. Channel Setting Function

7.1 channel_preset

int channel_preset(const channel* ch)	
Description	
Perform preset operation on a channel.	
Compatibility	2.0.39 and later.
Parameter description	
[in] **ch**	Target channel. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.2 channel_start

int channel_start(const channel* ch)	
Description	
Start channel operation.	
Compatibility	2.0.39 and later.
Parameter description	
[in] **ch**	Target channel. Refer to the channel structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.3 channel_power_state

int channel_power_state (const channel* ch, uint8_t state)	
Description	
Set channel power state.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] state	Power state. Values: POWER_ON / POWER_OFF / POWER_SAVING.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.4 channel_config_lo_mode

int channel_config_lo_mode (channel* ch, uint8_t mode)	
Description	
Configure channel LO mode.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] mode	Local oscillator mode, values: LOMODE_AUTO / LOMODE_HIGHSPEED / LOMODE_LOWSPUR / LOMODE_LOWPHASENOISE.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.5 channel_query_lo_mode

int channel_query_lo_mode (**channel* ch,** **uint8_t* mode_o** **)**	
Description	
Query channel LO mode.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] mode_o	Returns local oscillator mode, values: LOMODE_AUTO / LOMODE_HIGHSPEED / LOMODE_LOWSPUR / LOMODE_LOWPHASENOISE.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.6 channel_config_trigger

int channel_config_trigger(**channel* ch,** **int16_t strm,** **uint8_t ch_action** **)**	
Description	
Configure channel trigger parameters.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] strm	Stream index; currently fixed to stream0.
[in] ch_action	Channel action: TRIGGER_ACTION_HOP / TRIGGER_ACTION_SWEEP
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.

Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.7 channel_query_trigger

int channel_query_trigger(**channel* ch,** **int16_t* strm_o,** **uint8_t* ch_action_o** **)**	
Description	
Query channel trigger configuration.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] strm_o	Returns the index of the actually bound stream (currently 0).
[out]	Returns the channel action; the number of actions is specified by stream_trigger::response_count returned by tx_query_stream().
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.8 channel_bus_trigger

int channel_bus_trigger(**const channel* ch,** **uint32_t stream_mask** **)**	
Description	
Configure bus trigger.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channels. Refer to the channel structure definition for details.

[in] stream_mask	stream (stream index mask, currently inactive and can only be configured as 0)
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.9 channel_config_port

int channel_config_port (**channel* ch,** **uint16_t port** **)**	
Description	
Set RF output port.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] port	Port number: 0 for the standard output port, 1 for the high-power port.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

7.10 channel_query_port

int channel_query_port (**channel* ch,** **uint16_t* port_o** **)**	
Description	
Query RF output port.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.

[out] **port_o**	Return port number: 0 for the standard output port, 1 for the high-power port.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

7.11 channel_tx_query_capabilities

```
int channel_tx_query_capabilities(
    channel* ch,
    tx_channel_capabilities* capabilities
)
```

Description	
Query TX channel capabilities.	
Compatibility	2.0.39 and later.
Parameter description	
[in] **ch**	Target channel. Refer to the channel structure definition for details.
[out] **capabilities**	Returns TX channel capabilities. Refer to the tx_channel_capabilities structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

7.12 channel_config_amplitude_compensation

```
int channel_config_amplitude_compensation(
    const channel* ch,
    uint8_t  enable,
    const double* freq,
    const float* amplitude,
    uint32_t points
)
```

Description	
Configure amplitude compensation.	
Compatibility	2.0.39 and later.
Parameter description	
[in] **ch**	Channel handle. Refer to the channel structure definition for details.

[in] enable	Enable: 0 disable compensation, 1 enable compensation.
[in] freq	Frequency array.
[in] amplitude	Amplitude compensation value array.
[in] points	Number of frequency-response compensation points.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

8. Transmission Setting Functions

8.1 tx_config_output

int tx_config_output (**channel* ch,** **uint8_t state_rfout,** **uint8_t state_mod** **)**	
Description	
Configure channel output state (RFOUT / MOD).	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] state_rfout	RF output enable; values: STATE_OFF / STATE_ON.
[in] state_mod	Modulation output enable; values: STATE_OFF / STATE_ON.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

8.2 tx_query_output

int tx_query_output(**channel* ch,** **uint8_t* rfout_o,** **uint8_t* mod_o** **)**	
Description	
Query channel output state (RFOUT / MOD).	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] rfout_o	Return RF output state, values: STATE_OFF / STATE_ON

[out] mod_o	Return modulation output state, values: STATE_OFF / STATE_ON.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

8.3 tx_config_ffm

int tx_config_ffm(

channel* ch,
double fc,
float level

)

Description	
Configure channel frequency and amplitude parameters (FFM).	
Compatibility	1.0.1 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] fc	Center frequency (Hz).
[in] level	Output level (dBm).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
constexpr int16_t strm = 0;

device_info usb_devices[MAXCHANNELS]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);
uint16_t chs = usb_devices[dnum].chn;

device_info dinfo{};
channel ch[MAXCHANNELS]{};
status = device_open_usb(dnum, chs, &device, ch, &dinfo);
```

```
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(&ch[0], fc, level);

tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
setting.fc_offset = 0.0;
setting.phase_offset = 0.0f;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(&ch[0], strm, &setting, &trg);

status = channel_config_trigger(&ch[0], strm, TRIGGER_ACTION_HOP);

status = tx_config_output(&ch[0], STATE_ON, STATE_OFF);

status = channel_start(&ch[0]);

status = channel_bus_trigger(&ch[0], 1U << strm);

std::this_thread::sleep_for(std::chrono::seconds(1));

return device_close(&device);
```

8.4 tx_query_ffm

```
int tx_query_ffm(
    channel* ch,
    double* fc_o,
    float* level_o
)
```

Description	
Query channel frequency and amplitude parameters (FFM).	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] fc_o	Return center frequency (Hz).
[out] level_o	Return output level (dBm).

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

8.5 tx_config_fscan

```
int tx_config_fscan(
    channel* ch,
    double start,
    double stop,
    double step,
    float level,
    float dwell
)
```

Description	
Configure channel frequency scan parameters (FSCAN). With fixed power level, increment frequency from start to stop by step, dwelling at each frequency point for dwell.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] start	Scan start frequency (Hz).
[in] stop	Scan stop frequency (Hz).
[in] step	Scan step frequency (Hz).
[in] level	Output level (dBm).
[in] dwell	Dwell time per frequency point (s).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
device_info usb_devices[8]{};
uint32_t device_count = 0;
```

```
status = device_list_usb(usb_devices, &device_count);

device_info dinfo{};
channel ch[MAXCHANNELS]{};
uint16_t chn = usb_devices[dnum].chn;
status = device_open_usb(dnum, chn, &device, ch, &dinfo);

double start = 1.0e9;
double stop = 2.0e9;
double step = 100e6;
float level = 0.0f;
float dwell = 1e-4f;
constexpr uint32_t fscan_points = 11;
status = tx_config_fscan(ch, start, stop, step, level, dwell);

constexpr int16_t strm = 0;
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
trg.response_count = -1;
status = tx_config_stream(ch, strm, &setting, &trg);
status = channel_config_trigger(ch, strm, TRIGGER_ACTION_SWEEP);

device_trigger_out trigger_out{};
trigger_out.state = STATE_ON;
trigger_out.source = CHANNEL0;
trigger_out.negative_pulse = TRIGGER_PULSE_POSITIVE;
trigger_out.recounter = fscan_points;
status = device_config_trigger_out(&device, &trigger_out);

status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_bus_trigger(ch, 1U << strm);
std::this_thread::sleep_for(std::chrono::seconds(1));
status = device_close(&device);
return 0;
```

8.6 tx_query_fscan

int tx_query_fscan(channel* ch, double* start_o, double* stop_o, double* step_o, float* level_o, float* dwell_o)	
Description	
Query channel frequency scan parameters (FSCAN).	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] start_o	Return scan start frequency (Hz).
[out] stop_o	Return scan stop frequency (Hz).
[out] step_o	Return scan step frequency (Hz).
[out] level_o	Return output level (dBm).
[out] dwell_o	Return dwell time per frequency point (s).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_fscan function.

8.7 tx_config_lscan

int tx_config_lscan(channel* ch, double fc, float level_start, float level_stop, float level_step, float dwell)
Description
Configure channel amplitude scan parameters (LSCAN). With fixed center frequency fc, increment amplitude from level_start to level_stop by

level_step, dwelling at each amplitude point for dwell.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel definition structure.
[in] fc	Center frequency (Hz).
[in] level_start	Scan start amplitude (dBm).
[in] level_stop	Scan stop amplitude (dBm).
[in] level_step	Scan amplitude step (dB).
[in] dwell	Dwell time per amplitude point (s).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
constexpr uint32_t lscan_points = 11;
device_info usb_devices[8]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);

device_info dinfo{};
channel ch[MAXCHANNELS]{};
uint16_t chn = usb_devices[dnum].chn;
status = device_open_usb(dnum, chn, &device, ch, &dinfo);

status = tx_config_lscan(ch, 3.1e9, -20.0f, 0.0f, 2.0f, 1e-3f);

constexpr int16_t strm = 0;
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
trg.response_count = -1;
status = tx_config_stream(ch, strm, &setting, &trg);

status = channel_config_trigger(ch, strm, TRIGGER_ACTION_SWEEP);
```

```
device_trigger_out trigger_out{};
trigger_out.state = STATE_ON;
trigger_out.source = CHANNEL0;
trigger_out.negative_pulse = TRIGGER_PULSE_POSITIVE;
trigger_out.recounter = lscan_points;
status = device_config_trigger_out(&device, &trigger_out);

status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_bus_trigger(ch, 1U << strm);
std::this_thread::sleep_for(std::chrono::seconds(1));
status = device_close(&device);
return 0;
```

8.8 tx_query_lscan

int tx_query_lscan(**channel* ch,** **double* fc_o,** **float* level_start_o,** **float* level_stop_o,** **float* level_step_o,** **float* dwell_o** **)**	
Description	
Query channel amplitude scan parameters (LSCAN).	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[out] fc_o	Return center frequency (Hz).
[out] level_start_o	Return scan start amplitude (dBm).
[out] level_stop_o	Return scan stop amplitude (dBm).
[out] level_step_o	Return scan amplitude step (dB).
[out] dwell_o	Return dwell time per amplitude point (s).
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_lscan function.

8.9 tx_config_mscan

```
int tx_config_mscan(
    channel* ch,
    const double fc[],
    const float level[],
    const float dwell[],
    int16_t n
)
```

Description	
Configure channel multi-point scan parameters (MSCAN). Scan in array-index order from point 0 to point n-1; each point uses the corresponding fc[i] and level[i].	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.
[in] fc[]	Frequency array, length n.
[in] level[]	Amplitude array, length n.
[in] dwell[]	Dwell time array (s), length n
[in] n	Number of scan points
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

Example

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
constexpr uint32_t mscan_points = 4;
device_info usb_devices[8]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);

device_info dinfo{};
channel ch[MAXCHANNELS]{};
uint16_t chn = usb_devices[dnum].chn;
status = device_open_usb(dnum, chn, &device, ch, &dinfo);

double fc[] = {1e9, 2e9, 3e9, 4e9};
float level[] = {-30.0f, -20.0f, -10.0f, 0.0f};
```

```
float dwell[] = {1e-3f, 10e-3f, 20e-3f, 30e-3f};
status = tx_config_mscan(ch, fc, level, dwell, mscan_points);

constexpr int16_t strm = 0;
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_CW;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
trg.response_count = -1;
status = tx_config_stream(ch, strm, &setting, &trg);

status = channel_config_trigger(ch, strm, TRIGGER_ACTION_SWEEP);

device_trigger_out trigger_out{};
trigger_out.state = STATE_ON;
trigger_out.source = CHANNEL0;
trigger_out.negative_pulse = TRIGGER_PULSE_POSITIVE;
trigger_out.recounter = mscan_points;
status = device_config_trigger_out(&device, &trigger_out);

status = tx_config_output(ch, STATE_ON, STATE_OFF);
status = channel_start(ch);
status = channel_bus_trigger(ch, 1U << strm);
std::this_thread::sleep_for(std::chrono::seconds(1));
status = device_close(&device);
return 0;
```

8.10 tx_query_mscan

```
int tx_query_mscan(
    channel* ch,
    double fc_o[],
    float level_o[],
    float dwell_o[],
    int16_t* n
)
```

Description	
Query channel multi-point scan parameters (MSCAN). Return the currently configured scan sequence in array-index order.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Target channel. Refer to the channel structure definition for details.

[out] fc_o	Return frequency array, length n.
[out] level_o	Return amplitude array, length n.
[out] dwell_o	Return dwell time array (s), length n.
[out] n_o	Return number of scan points.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_mscan function.

8.11 tx_clear_waveform

int tx_clear_waveform(void device)**	
Description	
Clear all waveform data in the device.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_download_waveform function.

8.12 tx_download_waveform

int tx_download_waveform(**void** device,** **int16_t iq[],** **uint32_t points,** **uint16_t* waveform_o** **)**	
Description	
Download waveform data to the device. Only downloading waveforms in the order waveform = 0, 1, 2, 3... is supported.	
Compatibility	2.0.39 and later.
Parameter description	
[in] device	Device handle.

[in] iq	IQ date.
[in] points	Number of IQ points.
[out] waveform_o	Return waveform index.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.

Example

```
constexpr double PI = 3.14159265358979323846;

struct wave_header {
uint16_t audio_format;
uint16_t channels;
uint32_t sample_rate;
uint32_t byte_rate;
uint16_t block_align;
uint16_t bits_per_sample;
};

int get_wave(uint32_t& sample_rate, std::vector<int16_t>& samples);
std::vector<int16_t> sim_generatewaveform_sine();

int main(){
int status;
void* device = nullptr;

constexpr uint8_t dnum = 0;
constexpr int16_t strm = 0;

device_info usb_devices[MAXCHANNELS]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);
uint16_t chs = usb_devices[dnum].chn;

device_info dinfo{};
channel ch[MAXCHANNELS]{};

uint32_t samplerate = 125e6;
std::vector<int16_t> data = sim_generatewaveform_sine();

status = device_open_usb(dnum, chs, &device, ch, &dinfo);
```

```
double fc = 0.2e9;
float level = 0.0f;
status = tx_config_ffm(&ch[0], fc, level);

uint16_t waveid = 0;
status = tx_clear_waveform(&device);
status = tx_download_waveform(&device, data.data(), data.size() / 2, &waveid);

int16_t waveform[] = {static_cast<int16_t>(waveid)};
int32_t repeat[] = {-1};
double sample_rate[] = {static_cast<double>(samplerate)};

tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_PLAYBACK;
setting.fc_offset = 0.0;
setting.phase_offset = 0.0f;
setting.waveforms = 1;
setting.waveform = waveform;
setting.repeat = repeat;
setting.playback_srate = sample_rate;

stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(&ch[0], strm, &setting, &trg);

status = channel_config_trigger(&ch[0], strm, TRIGGER_ACTION_HOP);

status = tx_config_output(&ch[0], STATE_ON, STATE_ON);

status = channel_start(&ch[0]);

status = channel_bus_trigger(&ch[0], 1U << strm);

std::this_thread::sleep_for(std::chrono::seconds(1));
return device_close(&device);
}
std::vector<int16_t> sim_generatewaveform_sine()
{
constexpr uint32_t TOTAL_INT16_POINTS = 5000000;
constexpr uint32_t IQ_POINTS = TOTAL_INT16_POINTS / 2;
constexpr int16_t AMP = 30000;
constexpr uint32_t CYCLES = 25000;
std::vector<int16_t> iq;
```

```
iq.reserve(TOTAL_INT16_POINTS);
for (uint32_t n = 0; n < IQ_POINTS; ++n) {
double phase = 2.0 * PI * CYCLES * n / IQ_POINTS;
iq.push_back(static_cast<int16_t>(AMP * std::cos(phase)));
iq.push_back(static_cast<int16_t>(AMP * std::sin(phase)));
}
return iq;
}
int get_wave(uint32_t& sample_rate, std::vector<int16_t>& samples)
{
std::ifstream file("input.wav", std::ios::binary | std::ios::ate);
if (!file) {
std::cerr << "Cannot open input.wav" << std::endl;
return -1;
}

const std::streamoff file_size = file.tellg();
file.seekg(0);
char riff_header[12];
if (!file.read(riff_header, sizeof(riff_header)) || std::strncmp(riff_header, "RIFF", 4) != 0 ||
std::strncmp(riff_header + 8, "WAVE", 4) != 0) {
std::cerr << "input.wav is not a valid WAV file" << std::endl;
return -1;
}

wave_header header{};
static_assert(sizeof(wave_header) == 16, "Unexpected PCM format header layout");
bool format_found = false;
std::streamoff data_offset = -1;
uint32_t data_bytes = 0;
char chunk_id[4];
uint32_t chunk_size = 0;
while (file.read(chunk_id, sizeof(chunk_id)) &&
file.read(reinterpret_cast<char*>(&chunk_size), sizeof(chunk_size))) {
const std::streamoff chunk_offset = file.tellg();
if (chunk_size > file_size - chunk_offset) {
std::cerr << "input.wav contains an incomplete chunk" << std::endl;
return -1;
}

if (std::strncmp(chunk_id, "fmt ", 4) == 0 && !format_found) {
if (chunk_size < sizeof(header) || !file.read(reinterpret_cast<char*>(&header),
sizeof(header))) {
std::cerr << "Invalid input.wav format chunk" << std::endl;
```

```
return -1;
}
format_found = true;
} else if (std::strncmp(chunk_id, "data", 4) == 0 && data_offset < 0) {
data_offset = chunk_offset;
data_bytes = chunk_size;
}

if (format_found && data_offset >= 0)
break;

file.seekg(chunk_offset + chunk_size);
if ((chunk_size & 1U) != 0 && file.peek() == 0)
file.get();
}

if (!format_found || data_offset < 0) {
std::cerr << "Cannot find fmt/data chunks in input.wav" << std::endl;
return -1;
}
if (header.audio_format != 1 || header.channels != 2 || header.bits_per_sample != 16 ||
header.sample_rate == 0) {
std::cerr << "input.wav must be stereo 16-bit PCM with a valid sample rate" << std::endl;
return -1;
}

constexpr uint32_t iq_bytes = 2 * sizeof(int16_t);
const uint32_t valid_bytes = data_bytes - data_bytes % iq_bytes;
if (valid_bytes == 0) {
std::cerr << "input.wav contains no complete IQ samples" << std::endl;
return -1;
}

std::vector<int16_t> wave_samples(valid_bytes / sizeof(int16_t));
file.seekg(data_offset);
if (!file.read(reinterpret_cast<char*>(wave_samples.data()),
static_cast<std::streamsize>(valid_bytes))) {
std::cerr << "Failed to read waveform data from input.wav" << std::endl;
return -1;
}

samples.swap(wave_samples);
sample_rate = header.sample_rate;
return 0;}
```

8.13 tx_send_stream

int tx_send_stream(**channel* ch,** **int16_t strm,** **int16_t iq[],** **uint32_t points** **)**	
Description	
Send IQ data to the specified channel for real-time playback.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Specified channel. Refer to the channel structure definition for details.
[in] strm	stream index (currently only strm0 is available).
[in] iq	IQ data.
[in] points	Number of IQ points.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	

```
constexpr double PI = 3.14159265358979323846;

std::vector<int16_t> sim_generatewaveform_sine();

int main()
{
int status;
void* device = nullptr;

constexpr uint8_t dnum = 0;
constexpr int16_t strm = 0;

device_info usb_devices[MAXCHANNELS]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);
uint16_t chs = usb_devices[dnum].chn;

device_info dinfo{};
```

```
channel ch[MAXCHANNELS]{};

double sample_rate = 50e6;
std::vector<int16_t> data = sim_generatewaveform_sine();
status = device_open_usb(dnum, chs, &device, ch, &dinfo);
double fc = 1.0e9;
float level = 0.0f;
status = tx_config_ffm(&ch[0], fc, level);
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_REALTIME;
setting.fc_offset = 0.0;
setting.phase_offset = 0.0f;
setting.realtime_srate = sample_rate;

stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(&ch[0], strm, &setting, &trg);
status = channel_config_trigger(&ch[0], strm, TRIGGER_ACTION_HOP);
status = tx_config_output(&ch[0], STATE_ON, STATE_ON);
status = channel_start(&ch[0]);
status = channel_bus_trigger(&ch[0], 1U << strm);
for (int i = 0; i < 100000; ++i)
status = tx_send_stream(&ch[0], strm, data.data(), static_cast<uint32_t>(data.size() / 2));

return device_close(&device);
}
std::vector<int16_t> sim_generatewaveform_sine()
{
constexpr uint32_t TOTAL_INT16_POINTS = 5000000;
constexpr uint32_t IQ_POINTS = TOTAL_INT16_POINTS / 2;
constexpr int16_t AMP = 30000;
constexpr uint32_t CYCLES = 250000;
std::vector<int16_t> iq;
iq.reserve(TOTAL_INT16_POINTS);
for (uint32_t n = 0; n < IQ_POINTS; ++n) {
double phase = 2.0 * PI * CYCLES * n / IQ_POINTS;
iq.push_back(static_cast<int16_t>(AMP * std::cos(phase)));
iq.push_back(static_cast<int16_t>(AMP * std::sin(phase)));
}
return iq;
}
```

8.14 tx_config_stream

int tx_config_stream(const channel* ch, int16_t strm, const tx_stream* setting, const stream_trigger* trg)	
Description	
Configure TX stream.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Channel handle. Refer to the channel structure definition for details.
[in] strm	stream index (currently only strm0 is available).
[in] setting	TX stream service configuration. Refer to the tx_stream structure definition for details.
[in] trg	Stream trigger configuration; cached by the Host after successful configuration. Refer to the stream_trigger structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

8.15 tx_query_stream

int tx_query_stream(const channel* ch, int16_t strm, tx_stream* setting_o, stream_trigger* trg_o)	
Description	
Query TX stream configuration.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ch	Channel handle. Refer to the channel structure definition for details.

[in] strm	stream index (currently only strm0 is available)
[out] setting_o	Return TX stream service configuration. Refer to the tx_stream structure definition for details.
[out] trg_o	Returns the stream trigger configuration cached by the Host; the fmt pointer remains valid until the next configuration, preset, or device close operation. Refer to the stream_trigger structure definition for details.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the tx_config_ffm function.

9. Utility Functions

9.1 device_epoch_to_utc

void device_epoch_to_utc(**uint64_t ns_sinceepoch,** **int16_t* year,** **int16_t* mon,** **int16_t* day,** **int16_t* hour,** **int16_t* min,** **int16_t* sec,** **int16_t*ms,** **int16_t* us,** **int16_t* ns** **)**	
Description	
Convert a Unix Epoch nanosecond timestamp to readable UTC time.	
Compatibility	2.0.39 and later.
Parameter description	
[in] ns_sinceepoch	Time elapsed since Epoch, in ns.
[out] year	Year, for example 2026.
[out] mon	Month, 1 to 12.
[out] day	Day, 1 to 31.
[out] hour	Hour, 0 to 23.
[out] min	Minute, 0 to 59.
[out] sec	Second, 0 to 59.
[out] ms	Millisecond, 0 to 99.
[out] us	Microsecond, 0 to 999.
[out] ns	Nanosecond, 0 to 999.
Return value	None.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of tx_send_stream function.

10. Modulation Configuration Functions

10.1 mod_open

int mod_open(**void** device,** **char version_o[50]** **)**	
Description	
Enables the modulation function. After the device is opened, pass the device information structure to this function for device verification, check whether the modulation library exists, register the function, create the device object, and enable the modulation function.	
Compatibility	2.0.39 and later
Parameter description	
[out] device	Returns the device handle. This handle is used to reference the target device in subsequent API calls.
[out] version_o[50]	Returns the version number of the modulation library.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_usb or device_open_eth.
Example	Refer to the relevant example of the mod_config_digitalmod function.

10.2 mod_close

int mod_close(**void** device** **)**	
Description	
Disable the modulation function.	
Compatibility	2.0.39 and later
Parameter description	
[out] device	Device handle. The handle becomes invalid after the device is closed.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function should only be called before the modulation program terminates. After execution, the modulation function will be disabled and all related resources will be

	released. To enable modulation again, mod_open must be called again.
Example	Refer to the relevant example of the mod_config_digitalmod function.

10.3 mod_init_digitalmod

void mod_init_digitalmod(**void** device,** **digitalmod_profile* digitalmod_profile_o** **)**	
Description	
Initialize the digital modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] digitalmod_profile_o	Digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function.

10.4 mod_init_am

void mod_init_am(**void** device,** **am_profile* am_profile_o** **)**	
Description	
Initialize the AM modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] am_profile_o	AM configuration structure. For details, refer to the definition of the am_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.

Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.5 mod_init_fm

```
void mod_init_fm(
    void** device,
    fm_profile* fm_profile_o
)
```

Description	
Initialize the FM modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] fm_profile_o	FM configuration structure. For details, refer to the definition of the fm_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.6 mod_init_pm

```
void mod_init_pm(
    void** device,
    pm_profile* pm_profile_o
)
```

Description	
Initialize the PM modulation configuration structure. The default values are 1 kHz sine-wave modulation, 1 rad phase deviation, and 0 rad initial phase.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] pm_profile_o	PM configuration structure. For details, refer to the definition of the fm_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.

Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.7 mod_init_dsss

void mod_init_dsss(**void** device,** **dsss_profile* dsss_profile_o** **)**	
Description	
Initialize the direct-sequence spread spectrum configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] dsss_profile_o	Direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.8 mod_init_pulse

void mod_init_pulse(**void** device,** **pulse_profile* pulse_profile_o** **)**	
Description	
Initialize the pulse modulation configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] pulse_profile_o	Pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.

Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.9 mod_init_multitone

```
void mod_init_multitone(
    void** device,
    multitone_profile* multitone_profile_o
)
```

Description	
Initialize the multitone configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] multitone_profile_o	Multitone configuration structure. For details, refer to the definition of the multitone_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.10 mod_init_stepsweep

```
void mod_init_stepsweep(
    void** device,
    stepsweep_profile* stepsweep_profile_o
)
```

Description	
Initialize the step sweep configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] stepsweep_profile_o	Step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.

Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.11 mod_init_rampsweep

```
void mod_init_rampsweep(
    void** device,
    rampsweep_profile* rampsweep_profile_o
)
```

Description	
Initialize the ramp sweep configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] rampsweep_profile_o	Ramp sweep configuration structure. For details, refer to the definition of the rampsweep_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.12 mod_init_ofdm

```
void mod_init_ofdm (
    void** device,
    ofdm_profile* ofdm_profile_o
)
```

Description	
Initialize the OFDM configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] ofdm_profile_o	OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.

Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.13 mod_init_awgn

```
void mod_init_awgn (
    void** device,
    awgn_profile* awgn_profile_o
)
```

Description	
Initialize the Gaussian white noise configuration structure and assign default values to all parameters in the structure.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] awgn_profile_o	Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.14 mod_config_digitalmod

```
int mod_config_digitalmod(
    void** device,
    const digitalmod_profile* digitalmod_profile_in
    digitalmod_profile* digitalmod_profile_o
)
```

Description	
Configures the digital modulation structure parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.

[in] digitalmod_profile_in	Input digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.
[out] digitalmod_profile_o	Output digital modulation configuration structure. For details, refer to the definition of the digitalmod_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_open has been successfully executed.
Example	

```
int status;
void* device = nullptr;
constexpr uint8_t dnum = 0;
constexpr int16_t strm = 0;
device_info usb_devices[8]{};
uint32_t device_count = 0;
status = device_list_usb(usb_devices, &device_count);

device_info dinfo{};
channel ch[MAXCHANNELS]{};
uint16_t chs = usb_devices[dnum].chn;
status = device_open_usb(dnum, chs, &device, ch, &dinfo);

char mod_version[50]{};
status = mod_open(&device, mod_version);
std::cout << status;
status = tx_config_ffm(ch, 3.1e9, 0.0f);

digitalmod_profile digital{};
digitalmod_profile digital_applied{};
mod_init_digitalmod(&device, &digital);
digital.digitalmod_type = DigitalModType_QAM64;
digital.symbolrate = 62.5e6;
digital.fskdeviation = 0.0;
digital.pn = 15;
digital.sequenceseed = 23;
digital.filter_type = FilterType_RootRaisedCosine;
digital.filter_symbols = 16;
digital.sps = 2;
digital.alpha = 0.35;
status = mod_config_digitalmod(&device, &digital, &digital_applied);
```

```
int16_t* generated_iq = nullptr;
int32_t generated_iq_points = 0;
double samplerate = 0.0;
status = mod_generate_stream(&device, 1 << digital_applied.pn, &generated_iq,
&generated_iq_points, &samplerate);
std::vector<int16_t> data(generated_iq, generated_iq + generated_iq_points);

uint16_t waveid = 0;
status = tx_clear_waveform(&device);
status = tx_download_waveform(&device, data.data(), data.size() / 2, &waveid);

int16_t waveform[] = {static_cast<int16_t>(waveid)};
int32_t repeat[] = {-1};
double srate[] = {samplerate};
tx_stream setting{};
setting.state = STATE_ON;
setting.mode = TX_PLAYBACK;
setting.waveforms = 1;
setting.waveform = waveform;
setting.repeat = repeat;
setting.playback_srate = srate;
stream_trigger trg{};
trg.source = TRIGGER_SOURCE_BUS;
status = tx_config_stream(ch, strm, &setting, &trg);

status = channel_config_trigger(ch, strm, TRIGGER_ACTION_HOP);

status = tx_config_output(ch, STATE_ON, STATE_ON);
status = channel_start(ch);
status = channel_bus_trigger(ch, 1U << strm);
std::this_thread::sleep_for(std::chrono::seconds(1));

status = channel_stop(ch);
status = mod_close(&device);
status = device_close(&device);
return 0;
```

10.15 mod_config_am

int mod_config_am(void** device, const am_profile* am_profile_in am_profile* am_profile_o)	
Description	
Configures the AM modulation parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] am_profile_in	Input AM modulation configuration structure. For details, refer to the definition of the am_profile structure.
[out] am_profile_o	Output AM modulation configuration structure. For details, refer to the definition of the am_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_am has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.16 mod_config_fm

int mod_config_fm(void** device, const fm_profile* fm_profile_in fm_profile* fm_profile_o)	
Description	
Configures the FM modulation parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] fm_profile_in	Input FM modulation configuration structure. For details, refer to the definition of the fm_profile structure.

[out] fm_profile_o	Input FM modulation configuration structure. For details, refer to the definition of the fm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_fm has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.17 mod_config_pm

```
int mod_config_pm(
    void** device,
    const pm_profile* pm_profile_in
    pm_profile* pm_profile_o
)
```

Description	
Configures the PM modulation parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.37 and later
Parameter description	
[in] device	Device handle.
[in] pm_profile_in	Input PM modulation configuration structure. For details, refer to the definition of the pm_profile structure.
[out] pm_profile_o	Input PM modulation configuration structure. For details, refer to the definition of the pm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_fm has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.18 mod_config_dsss

int mod_config_dsss(void** device, const dsss_profile* dsss_profile_in, dsss_profile* dsss_profile_o)	
Description	
Configures the direct-sequence spread spectrum parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] dsss_profile_in	Input direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.
[out] dsss_profile_o	Output direct-sequence spread spectrum configuration structure. For details, refer to the definition of the dsss_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_dass has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.19 mod_config_pulse

int mod_config_pulse(void** device, const pulse_profile* pulse_profile_in pulse_profile* pulse_profile_o)	
Description	
Configures the pulse modulation parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.

[in] pulse_profile_in	Input pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.
[out] pulse_profile_o	Output pulse modulation configuration structure. For details, refer to the definition of the pulse_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_pulse has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.20 mod_config_multitone

```
int mod_config_multitone(
    void** device,
    const multitone_profile* multitone_profile_in
    multitone_profile* multitone_profile_o
)
```

Description	
Configures the multi-tone parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] multitone_profile_in	Input multitone modulation configuration structure. For details, refer to the definition of the multitone_profile structure.
[out] multitone_profile_o	Output multitone modulation configuration structure. For details, refer to the definition of the multitone_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_multitone has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.21 mod_config_stepsweep

int mod_config_stepsweep(**void** device,** **const stepsweep_profile* stepsweep_profile_in** **stepsweep_profile* stepsweep_profile_o** **)**	
Description	
Configures the step sweep parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] stepsweep_profile_in	Input step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.
[out] stepsweep_profile_o	Output step sweep configuration structure. For details, refer to the definition of the stepsweep_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_stepsweep has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.22 mod_config_rampsweep

int mod_config_rampsweep(**void** device,** **const rampsweep_profile* rampsweep_profile_in** **rampsweep_profile* rampsweep_profile_o** **)**	
Description	
Configures the ramp sweep parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[out] rampsweep_profile_o	Ramp sweep configuration structure. For details, refer to the definition of the rampsweep_profile structure.

Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_rampsweep has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.23 mod_config_ofdm

```
int mod_config_ofdm (
    void** device,
    const ofdm_profile* ofdm_profile_in
    ofdm_profile* ofdm_profile_o
)
```

Description	
Configures the orthogonal frequency-division multiplexing (OFDM) parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] ofdm_profile_o	Input OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.
[out] ofdm_profile_o	Output OFDM configuration structure. For details, refer to the definition of the ofdm_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_ofdm has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.24 mod_config_awgn

int mod_config_awgn (void** device, const awgn_profile* awgn_profile_in awgn_profile* awgn_profile_o)	
Description	
Configures the additive white Gaussian noise parameters. If the input configuration is invalid, the output configuration structure will be updated with valid parameter values.	
Compatibility	2.0.39 and later
Parameter description	
[in] device	Device handle.
[in] awgn_profile_in	Input Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.
[out] awgn_profile_o	Output Gaussian white noise configuration structure. For details, refer to the definition of the awgn_profile structure.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after mod_init_awgn has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function. Replace the profile/init/config function with the corresponding modulation type.

10.25 mod_generate_stream

int mod_generate_stream (void** device, int32_t symbol_length, int16_t** iq_o, int32_t* length_o, double* sample_rate_o)	
Description	
After enabling the modulation function, call the initialization function to assign default values, and then use the configuration structure functions to apply parameters and generate signal data.	
Compatibility	2.0.39 and later
Parameter description	

[in] device	Device handle.
[in] symbol_length	Symbol length, used for digital modulation and direct-sequence spread spectrum modulation.
[out] iq_o	Output signal data.
[out] length_o	Output data length.
[out] sample_rate_o	Sampling rate used for generating the current data.
Return value	0: Success; Non-zero: Error code. Refer to the Appendix 1 for details.
Calling constraints	This function must be called after device_open_* has been successfully executed.
Example	Refer to the relevant example of the mod_config_digitalmod function.

11. Structure Variables

11.1 eth_setting

eth_setting	
uint8_t **is_ipv6**	ETH protocol type (0: IPv4, 1: IPv6).
uint8_t **ip_address[16]**	IP address of the ETH interface. For IPv4, use the first 4 bytes.
uint16_t **remote_port**	Remote port of the ETH interface.
int32_t **read_timeout**	ETH interface read timeout, in ms.
void* **extension1**	Reserved for extension.
void* **extension2**	Reserved for extension.

11.2 gnss_setting

gnss_setting	
uint8_t **antenna**	Antenna path selection. 0 for internal antenna, 1 for external antenna.
int8_t **constellations**	Constellation enable mask; values: CONSTELLATION_GPS/CONSTELLATION_GLONASS/ CONSTELLATION_BEIDOU / CONSTELLATION_GALILEO, bitwise OR.
uint8_t **state_pps_out**	PPS output enable; values: STATE_OFF / STATE_ON
float **pps_x**	PPS frequency (Hz)
float **pps_delay**	PPS delay (s)
void* **extension1**	Reserved for extension.
void* **extension2**	Reserved for extension.

11.3 gnss_data

gnss_data	
uint8_t **is_locked**	GNSS lock state: 0 unlocked, 1 locked.
uint8_t **sat_nums**	Current number of positioning satellites.
uint8_t **snr_max**	Maximum satellite SNR.
uint8_t **snr_avg**	Average satellite SNR.
uint8_t **snr_min**	Minimum satellite SNR.
float **latitude**	Latitude. North is positive; south is negative.
float **longitude**	Longitude. East is positive; west is negative.
float **altitude**	Altitude, in m.
uint64_t **ns_sinceepoch**	Time elapsed since Epoch, in ns.

void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.4 device_info

device_info	
uint16_t model	Device model.
char model_name[16]	Device model string
uint8_t chn	Numbers of channels.
uint64_t uid_l64	Low 64-bit device serial number.
uint32_t uid_h32	High 32-bit device serial number.
uint8_t versions	Number of version entries.
char version_name[16][16]	Version string array.
uint8_t interfaces	Number of interfaces.
char interface_name[8][16]	Interface name array.
void* extension1	Reserved for extension.
void* extension2	Reserved for extension.

11.5 option

option	
uint16_t code	Option code. Values: OPTION_MEDIUM_POWER / OPTION_PULSE_DET / OPTION_EMI / OPTION_BW_100M_RX / OPTION_BW_320M_TX.
char description[64]	Option description.
uint8_t state	Option state: 0 disabled, 1 enabled.
uint32_t data[16]	Option auxiliary data.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.6 channel

channel	
void device**	Device handle to which the channel belongs.
uint16_t num	Channel number.
uint8_t type	Channel type. Values: CHANNELTYPE_RX / CHANNELTYPE_TX.
uint16_t max_streams	Maximum number of streams supported by the channel

void* extension1	Reserved for future expansion.
void* extension2	Reserved for future expansion.

11.7 stream_trigger

stream_trigger	
uint8_t source	Trigger source.
int64_t response_count	Number of trigger responses; -1 indicates unlimited responses, >= 0 indicates stopping after responding to a limited number of triggers
uint64_t triggerdelay	Trigger delay in points (currently not supported)
uint64_t pretrigger	Number of pre-trigger points (currently not supported)
uint64_t retrigger_interval	Retrigger interval in points (currently not supported)
uint64_t retrigger_count	Number of retriggers; internally generated by the device and not counted in response_count (currently not supported)
float trigger_level	Trigger threshold (currently not supported)
uint64_t safetime	Number of safety points (currently not supported)
float* fmt	MASK array (currently not supported)
uint64_t fmt_points	Number of mask points (currently not supported)
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.8 device_trigger_out

device_trigger_out	
uint8_t state	Trigger output enable. Values: STATE_OFF / STATE_ON.
uint8_t source	TX trigger-output source. Values: CHANNEL0 / CHANNEL1. Streams are not currently supported.
uint8_t negative_pulse	Negative-pulse enable: 0 positive pulse, 1 negative pulse.
uint32_t recounter	Generates one external trigger after N internal triggers.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.9 gain_setting

gain_setting	
float ref_level	Reference level, in dBm.
uint8_t state_manual_atten	Manual attenuation enable. Values:

	STATE_OFF / STATE_ON.
uint8_t atten	Manual attenuation value, in dB.
uint8_t amplifers	Gain configuration. Values: AMPLIFERS_AUTO / AMPLIFERS_PREAMP_ON_PATHGAIN_HIGH / AMPLIFERS_PREAMP_OFF_PATHGAIN_HIGH/ AMPLIFERS_PREAMP_OFF_PATHGAIN_MEDIUM / AMPLIFERS_PREAMP_OFF_PATHGAIN_LOW.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.10 agc_setting

agc_setting	
uint8_t state	AGC enable. Values: STATE_OFF / STATE_ON.
uint8_t mode	AGC mode. Value: AGC_MODE_AUTO.
float target_level	AGC target level, in dBm.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.11 stream

stream	
uint8_t state	Stream enables. Values: STATE_OFF / STATE_ON.
uint8_t state_output	Data output enable. Values: STATE_OFF / STATE_ON.
double samplerate	Sample rate, in Hz.
double fc_offset	Center-frequency offset, in Hz.
float phase_offset	Phase offset, in rad.
float scale	Amplitude scale.
uint8_t state_32bit_iq	32-bit IQ output enable. Values: STATE_OFF / STATE_ON.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.12 data_organization

data_organization	
uint8_t **point_size**	Bytes per point.
uint32_t **hops_in_sweep**	Number of hops in one sweep.
uint32_t **packets_in_hop**	Number of packets in one hop.
uint32_t **frames_in_packet**	Number of frames in one packet.
uint32_t **points_in_frame**	Number of points in one frame.
void* **extension1**	Reserved for extension
void* **extension2**	Reserved for extension

11.13 processer_det

processer_det	
uint8_t **state**	Detector processor enable. Values: STATE_OFF / STATE_ON.
uint8_t **state_output**	Data output enable. Values: STATE_OFF / STATE_ON.
double **rbw**	Resolution bandwidth, in Hz.
uint8_t **state_6db_rbw**	6 dB RBW enable. Values: STATE_OFF / STATE_ON.
uint8_t **trace_detector**	Trace detector mode. Values: DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS.
float **trace_detector_ratio**	Trace detector ratio.
void* **extension1**	Reserved for extension
void* **extension2**	Reserved for extension

11.14 processer_fft

processer_fft	
uint8_t **state**	FFT processor enable. Values: STATE_OFF / STATE_ON.
uint8_t **state_output_frame**	Frame data output enable. Values: STATE_OFF / STATE_ON.
uint8_t **state_output_psacn**	Scan (pscan) data output enable. Values:

	STATE_OFF / STATE_ON.
uint8_t state_output_bitmap	Bitmap data output enable. Values: STATE_OFF / STATE_ON.
double rbw	Resolution bandwidth, in Hz.
double vbw	Video bandwidth, in Hz.
uint8_t window	FFT window. Values: FFTWINDOW_NUTTALL / FFTWINDOW_FLATTOP / FFTWINDOW_LOWSIDELOBE / FFTWINDOW_GUASSIAN.
float overlap	Inter-frame overlap ratio.
uint8_t frame_detector	Frame detector mode. Values: DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS.
float frame_detector_ratio	Frame detector ratio.
uint8_t trace_detector	Trace detector mode. Values: DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS.
float trace_detector_ratio	Trace detector ratio.
uint8_t state_8bit_amplitude	8-bit amplitude output enable. Values: STATE_OFF / STATE_ON.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.15 aux_data

aux_data	
double fc	Current center frequency, in Hz.
double bin	Current frequency bin resolution, in Hz.
float scale	Amplitude scale.
float offset	Amplitude offset.
uint32_t hops_in_sweep	Number of hops in one sweep.
uint32_t packets_in_hop	Number of packets in one hop.
uint32_t frames_in_packet	Number of frames in one packet.

uint32_t points_in_frame	Number of points in one frame.
uint32_t sweepi	Current sweep index.
uint32_t hopi	Current hop index.
uint32_t packeti	Current packet index.
int32_t underrun	Data underrun count.
float max_power	Maximum power.
uint32_t max_power_index	Index corresponding to maximum power.
float temperature	Device temperature.
int32_t health	Device health state.
gnss_data gnss_data	GNSS information. Refer to the gnss_data structure definition for details.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.16 tx_stream

tx_stream	
uint8_t state	Enable. Values: STATE_OFF / STATE_ON.
uint8_t mode	Operating mode. Values: TX_CW / TX_REALTIME / TX_PLAYBACK.
double fc_offset	Frequency offset, in Hz.
float phase_offset	Phase offset, in rad.
double realtime_srate	Real-time playback sample rate, in Hz.
int16_t waveforms	Playback waveform count.
int16_t* waveform	Playback waveform[i], the waveform index in memory.
int32_t* repeat	Playback repeat[i], waveform repeat count.
double* playback_srate	Playback srate[i], waveform sample rate.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.17 tx_channel_capabilities

tx_channel_capabilities	
double freq_min	Minimum center frequency, in Hz.
double freq_max	Maximum center frequency, in Hz.
double freq_step_min	Minimum frequency step, in Hz.

double freq_step_max	Maximum frequency step, in Hz.
double bandwidth_min	Minimum bandwidth, in Hz.
double bandwidth_max	Maximum bandwidth, in Hz.
double level_min	Minimum output level, in dBm.
double level_max	Maximum output level, in dBm.
double level_step_min	Minimum level step, in dB.
double level_step_max	Maximum level step, in dB.
double dwelltime_min	Minimum dwell time, in s.
double dwelltime_max	Maximum dwell time, in s.
double realtime_srate_min	Minimum real-time playback sample rate, in Hz.
double realtime_srate_max	Maximum real-time playback sample rate, in Hz.
double playback_srate_min	Minimum playback sample rate, in Hz.
double playback_srate_max	Maximum playback sample rate, in Hz.
uint64_t memory_size	Waveform memory size, in bytes.
void* extension1	Reserved for extension
void* extension2	Reserved for extension

11.18 digitalmod_profile

digitalmod_profile	
uint8_t digitalmod_type	Digital modulation type. Refer to the digital modulation type macro definitions for valid values.
double symbolrate	Symbol rate, in Hz. Range: minimum sampling rate / samples per symbol ~ maximum sampling rate / samples per symbol.
double fskdeviation	FSK frequency deviation, in Hz, maximum is 15 × symbol rate.
int32_t pn	Pseudo-random sequence, range 4 to 32, recommended maximum not exceeding 24.
int32_t sequenceseed	Initial value of the pseudorandom sequence, 1 ~ 2147483647
uint8_t filter_type	Filter type. Values: MOD_FILTER_RECTANGULAR / MOD_FILTER_RAISED_COSINE / MOD_FILTER_ROOT_RAISED_COSINE / MOD_FILTER_GAUSSIAN / MOD_FILTER_HALF_SINE

int32_t filter_symbols	Number of filter symbols, must be an even number. Range: 2 to (sps × filter_symbols ≤ 400), default is 8.
int32_t sps	Samples per symbol, even number from 4 to 32, default is 4.
double alpha	Raised cosine / root raised cosine: 0.025 to 1 Gaussian filter: 0.15 to 2.5 The alpha parameter is only valid for raised cosine, root raised cosine, and Gaussian filtering.

11.19 am_profile

am_profile	
double rate	Modulation rate, in Hz.
double depth	Modulation depth, in %.
uint8_t shape	Modulation waveform. Value: MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.20 fm_profile

fm_profile	
double rate	Modulation rate, in Hz.
double deviation	Frequency deviation of the modulation, in Hz.
uint8_t shape	Modulation waveform. Value : MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.21 pm_profile

pm_profile	
double rate	Modulation rate, in Hz.
double phase_deviation	Maximum phase deviation, in rad, range: [0, 2*pi]
double init_phase	Initial phase, in rad; the configured value is normalized to [-pi, pi]
uint8_t shape	Modulation waveform. Available values: MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.22 dsss_profile

dsss_profile	
double symbolrate	Symbol rate, in Hz.
uint8_t digitalmod_type	Digital modulation type. Refer to the digital modulation type macro definitions for valid values.

int32_t code	Spreading code, PN value, recommended value: 5.
int32_t filter_symbols	Number of filter symbols, must be an even number. Range: 2 to (sps × filter_symbols ≤ 400), default is 8.
int32_t sps	Samples per symbol, even number from 4 to 32, default is 4.
double alpha	Raised cosine / root raised cosine: 0.025 to 1 Gaussian filter: 0.15 to 2.5 The alpha parameter is only valid for raised cosine, root raised cosine, and Gaussian filtering.
int32_t sequenceseed	Initial seed of the pseudo-random sequence, range: 1 to 2,147,483,647.
uint8_t filter_type	Filter type. Values: MOD_FILTER_RECTANGULAR / MOD_FILTER_RAISED_COSINE / MOD_FILTER_ROOT_RAISED_COSINE / MOD_FILTER_GAUSSIAN / MOD_FILTER_HALF_SINE

11.23 pulse_profile

pulse_profile	
double width	Pulse width, in seconds (s).
double period	Period, in seconds (s). The period must be greater than the pulse width.

11.24 multitone_profile

multitone_profile	
uint8_t phase_mode	Phase mode. Values: MOD_MULTITONE_PHASE_FIXED / MOD_MULTITONE_PHASE_RANDOM / MOD_MULTITONE_PHASE_PARABOLIC
uint32_t seed	Random seed for MOD_MULTITONE_PHASE_RANDOM mode; ignored in other modes
int32_t tonecount	Number of candidate tones, range: [2, MOD_MULTITONE_MAX_TONES]
double freqsapcing	Frequency spacing between adjacent tones, in Hz
int32_t generate_tones	Number of candidate tones to generate; set equal to tonecount to output all tones
const int32_t* generate_tone_idx	Indices of candidate tones to generate, range: [0, tonecount - 1]; can be nullptr when outputting all tones

11.25 stepsweep_profile

stepsweep_profile	
double center	Center frequency, in Hz.
double span	Span, in Hz.
int32_t points	Number of points.
double dwelltime	Dwell time, in seconds (s).

11.26 rampsweep_profile

rampsweep_profile	
double span	Span, in Hz.
double sweeptime	Sweep time, in seconds (s).
double period	Period, in seconds (s).

11.27 ofdm_profile

ofdm_profile	
double idleinterval	Idle time between transmit bursts, in seconds (s).
int32_t fftsize	Number of FFT points, supporting 16, 32, 64, 128, 256, 512, 1024, and 2048.Additionally, fftsize > guardbandcarriers_left + guardbandcarriers_right.
int32_t symbolcount	Number of symbols.
int32_t guardbandcarriers_left	Number of guard carriers on the left side.
int32_t guardbandcarriers_right	Number of guard carriers on the right side.
int32_t guardinterval	Guard interval, in %.
int32_t window_length	Window length, in %.
uint8_t windowed	Whether windowing is enabled.
uint8_t digitalmod_type	Digital modulation type. Refer to the digital modulation type macro definitions for valid values.
uint8_t nulldc	Whether DC is nulled.
double samplerate	Sampling rate: 195.3125e3 to 125e6, default is 20 MHz.

11.28 awgn_profile

awgn_profile	
double bandwidth	Bandwidth, in Hz
double length	Duration, in seconds

12. Enumeration Variables

12.1 mod_digitalmod_type

DigitalModType_BPSK	BPSK
DigitalModType_DBPSK	DBPSK
DigitalModType_QPSK	QPSK
DigitalModType_DQPSK	DQPSK
DigitalModType_OQPSK	OQPSK
DigitalModType_Pi4DQPSK	Pi4DPSK
DigitalModType_PSK8	8PSK
DigitalModType_D8PSK	D8PSK
DigitalModType_PSK16	16PSK
DigitalModType_QAM16	16QAM
DigitalModType_QAM64	64QAM
DigitalModType_QAM256	256QAM
DigitalModType_QAM1024	1024QAM
DigitalModType_ASK2	2ASK
DigitalModType_FSK2	2FSK
DigitalModType_FSK4	4FSK
DigitalModType_FSK8	8FSK
DigitalModType_FSK16	16FSK
DigitalModType_ASK4	4ASK
DigitalModType_ASK8	8ASK
DigitalModType_APSK16	16ASK
DigitalModType_QAM8	8QAM

12.2 mod_filter_type

FilterType_Rectangular	Rectangular filter
FilterType_RaisedCosine	Raised cosine filter
FilterType_RootRaisedCosine	Root raised cosine filter
FilterType_Gaussian	Gaussian filter
FilterType_HalfSine	Half-sine filter

12.3 mod_shape_type

ModShape_Sine	Sine wave
ModShape_Square	Square wave
ModShape_Triangle	Triangle wave
ModShape_Ramp	Sawtooth wave

13. Macro Definitions

13.1 Language

H2_LANGUAGE_ENGLISH	0	English
H2_LANGUAGE_CHINESE	1	Chinese

13.2 Option codes

OPTION_MEDIUM_POWER	10	Medium power option
OPTION_PULSE_DET	72	Pulse detection option
OPTION_EMI	73	EMI measurement option
OPTION_BW_100M_RX	50	100 MHz spectrum analyzer bandwidth option
OPTION_BW_320M_TX	53	320 MHz signal generator bandwidth option

13.3 Common state

AUTO	(-1)	Automatic
OFF	0	Off
ON	1	On
STATE_OFF	0	State off
STATE_ON	1	State on

13.4 Reference clock

REF_INT	0	Reference clock source internal
REF_EXT	1	Reference clock source external
REF_OUT_OFF	0	Reference clock output: off
REF_OUT_ON	1	Reference clock output: on
INTERNAL	0	Internal
EXTERNAL	1	External. Corrected from the previous duplicate value 0

13.5 GNSS constellation

CONSTELLATION_GPS	1	GPS constellation (bit0)
CONSTELLATION_GLONASS	2	GLONASS constellation (bit1)
CONSTELLATION_BEIDOU	4	BeiDou constellation (bit2)
CONSTELLATION_GALILEO	8	Galileo constellation (bit3)

13.6 Device interface type

INTERFACE_USB	0	USB interface
INTERFACE_ETH_1000M	1	Gigabit Ethernet interface
INTERFACE_ETH_10G	2	10G Ethernet interface
INTERFACE_ETH_25G	3	25G Ethernet interface
INTERFACE_ETH_40G	4	40G Ethernet interface
INTERFACE_ETH_100G	5	100G Ethernet interface

13.7 Channel common definitions

ALLCHANNEL	-1	All channels.
MAXCHANNELS	8	Maximum number of channels.
CHANNELTYPE_RX	0	Channel type: RX.
CHANNELTYPE_TX	1	Channel type: TX.
CHANNEL0	0	Channel index 0.
CHANNEL1	1	Channel index 1.
LOMODE_AUTO	0	LO mode: automatic.
LOMODE_HIGHSPEED	1	LO mode: high speed. Corrected from invalid literal 1s.
LOMODE_LOWSPUR	2	LO mode: low spur.
LOMODE_LOWPHASENOISE	3	LO mode: low phase noise.

13.8 Power definitions

USB_RAIL	0	Power rail: USB port.
RF_RAIL	1	Power rail: RF port.
POWER_STATE	1	Function selector: power state.
POWER_ON	0	Power state: on.
POWER_OFF	1	Power state: off.
POWER_SAVING	2	Power state: power saving.

13.9 RX gain and AGC

AMPLIFERS_AUTO	0	Gain: automatic.
AMPLIFERS_PREAMP_ON_PATHGAIN_HIGH	1	Gain: preamplifier on, high path gain.
AMPLIFERS_PREAMP_OFF_PATHGAIN_HIGH	2	Gain: preamplifier off, high path gain.
AMPLIFERS_PREAMP_OFF_PATHGAIN_MEDIUM	3	Gain: preamplifier off, medium path gain.
AMPLIFERS_PREAMP_OFF_PATHGAIN_LOW	4	Gain: preamplifier off, low path gain.
AGC_MODE_AUTO	0	AGC mode: automatic.

13.10 RX analyzer detectors

DETECTOR_MAX	0	Detector: maximum.
DETECTOR_MIN	1	Detector: minimum.
DETECTOR_AVG	2	Detector: average.
DETECTOR_RMS	3	Detector: RMS.
DETECTOR_QP	5	Detector: quasi-peak.
DETECTOR_MAXPOWER	6	Detector: maximum power.
DETECTOR_AUTOPEAK	7	Detector: automatic peak.
DETECTOR_BYPASS	8	Detector: bypass.

13.11 FFT window types

FFTWINDOW_NUTTALL	0	FFT window: Nuttall.
FFTWINDOW_FLATTOP	1	FFT window: FlatTop.
FFTWINDOW_LOWSIDELOBE	2	FFT window: low sidelobe.
FFTWINDOW_GUASSIAN	3	FFT window: Gaussian.

13.12 TX modes

TX_CW	0	TX mode: continuous wave.
TX_REALTIME	1	TX mode: real-time playback.
TX_PLAYBACK	2	TX mode: waveform playback.

13.13 Trigger Input Source

TRIGGER_SOURCE_BUS	0	Bus trigger.
TRIGGER_SOURCE_EXTERNAL_RISING_EDGE	1	External rising-edge trigger
TRIGGER_SOURCE_EXTERNAL_FALLING_EDGE	2	External falling-edge trigger
TRIGGER_SOURCE_XPPS	3	GNSS pulse-per-second trigger
TRIGGER_SOURCE_LEVEL	4	Level trigger, currently not supported
TRIGGER_SOURCE_FMT	5	Frequency mask trigger, currently not supported
TRIGGER_SOURCE_TIMEDOMAIN_GATE	6	Time-domain gate trigger, currently not supported

13.14 Trigger Output Source

TRIGGER_IN_ROUTE	2	Raw trigger event output

13.15 Trigger Output Pulse Polarity

TRIGGER_PULSE_POSITIVE	0	Positive pulse output
TRIGGER_PULSE_NEGATIVE	1	Negative pulse output

13.16 Trigger Channel Action

TRIGGER_ACTION_HOP	0	Executes one frequency hop.
TRIGGER_ACTION_SWEEP	1	Executes one complete sweep.

13.17 Digital Modulation Type

MOD_DIGITAL_BPSK	1	BPSK
MOD_DIGITAL_DBPSK	2	DBPSK
MOD_DIGITAL_QPSK	3	QPSK
MOD_DIGITAL_DQPSK	4	DQPSK
MOD_DIGITAL_OQPSK	5	OQPSK
MOD_DIGITAL_PI4_DQPSK	6	DQPSK
MOD_DIGITAL_PSK8	7	PSK8
MOD_DIGITAL_D8PSK	8	D8PSK
MOD_DIGITAL_PSK16	9	PSK16
MOD_DIGITAL_QAM16	10	QAM16
MOD_DIGITAL_QAM64	11	QAM64
MOD_DIGITAL_QAM256	12	QAM256
MOD_DIGITAL_QAM1024	13	QAM1024
MOD_DIGITAL_ASK2	14	ASk2
MOD_DIGITAL_FSK2	15	FSK2
MOD_DIGITAL_FSK4	16	FSK4
MOD_DIGITAL_FSK8	17	FSK8
MOD_DIGITAL_FSK16	18	FSK16
MOD_DIGITAL_ASK4	19	ASK4
MOD_DIGITAL_ASK8	20	ASK8
MOD_DIGITAL_APSK16	21	APSK16
MOD_DIGITAL_QAM8	22	QAM8

13.18 Filter Type

MOD_FILTER_RECTANGULAR	1	Rectangular Filter
MOD_FILTER_RAISED_COSINE	2	Raised Cosine Filter
MOD_FILTER_ROOT_RAISED_COSINE	3	Root Raised Cosine Filter
MOD_FILTER_GAUSSIAN	4	Gaussian Filter
MOD_FILTER_HALF_SINE	5	Half-Sine Filter

13.19 Modulation Wave

MOD_SHAPE_SINE	1	Sine Wave
MOD_SHAPE_SQUARE	2	Square Wave
MOD_SHAPE_TRIANGLE	3	Triangle Wave
MOD_SHAPE_RAMP	4	Sawtooth Wave

13.20 Multi-tone Phase Mode

MOD_MULTITONE_PHASE_FIXED	0	Use a fixed initial phase of 0 rad for all tones
MOD_MULTITONE_PHASE_RANDOM	1	Use a seed to generate reproducible random phases
MOD_MULTITONE_PHASE_PARABOLIC	2	Use a quadratic phase law to reduce the PAPR

Appendix 1: API Return Value Index

Error Code	Error/Warning Reason	Type[1]	Handling
0	No error	-	No action required; subsequent operations can proceed normally.
-1	Bus open error	Error	Check device power and data cable connections. Verify that the driver is correctly installed. After resolving the issue, call device_open_usb again to open the device.
-8	Bus transmission error	Error	Reconnect the device. If the issue persists, please contact official technical support.
-29	Transmit power calibration file missing [2]	Error	Check whether the transmit power calibration file is placed in the designated directory. After resolving the issue, call device_open_usb again to open the device.
-43	IQ calibration file missing [2]	Error	Check whether the IQ calibration file is placed in the designated directory. After resolving the issue, call device_open_usb again to open the device.
-47	Required option unavailable.	Error	Verify that the required option is installed, enabled, and not expired.
-601	Modulation license file missing	Error	Please place the xx._mod.lic file in the /CalFile/ folder.
-602	Modulation license file verification failed	Error	Usually due to incorrect license content; please contact official technical support.
-603	Device validation failed	Error	Device validation failed when creating the device object. Please check if the device is properly opened and contact official technical support.
-604	Object creation failed	Error	Object creation failed when creating the device object. Please check if the device is properly opened and contact official technical support.
-605	Object release failed	Error	Failed to release the device object. Please contact official technical support.
-606	Modulation type error	Error	When generating a signal, the input modulation type is incorrect. Please correct the modulation type, resend, and regenerate the signal.
-607	Modulation parameter error	Error	When generating a signal, the input modulation parameters are incorrect. Please correct the modulation parameters, resend, and regenerate the signal.
-608	Modulation library missing	Error	The modulation library is not found in the same path as the h2_api library. Please place genSignalWave.dll (Windows) or libgenSignalWave.so (Linux) in the same path as the h2_api library.

-609	Parameter cannot be configured	Error	The current parameter cannot be configured. The last available parameter has been applied.
10	Data transmission timeout	Warning	Wait 10 ms, then retry sending.
42	Using default transmit power calibration file	Warning	Does not affect device operation, but RF output power may deviate.
44	Using default IQ calibration file	Warning	Does not affect device operation, but accuracy may be affected.
98	Parameter out of range, clamped	Warning	The out-of-range parameters are automatically forced back into the device's valid range. The actual values at that time can be queried via the query function. This does not affect continued device operation.
99	Transmit power out of range; output accuracy may be affected.	Warning	The power level may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
15	System clock unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
16	AD/DA clock unlocked.	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
17	DLREF reference clock unlocked.	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
18	RF local oscillator unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.
19	IF local oscillator unlocked	Warning	The test results may be inaccurate, but this does not affect continued device operation. Please contact official technical support.

[1]. If the type is "Error," the issue must be resolved immediately and the device reopened; otherwise, the device cannot continue with subsequent operations. If the type is "Warning," the device can continue with subsequent operations without closing or reopening, but it is still recommended to handle specific return values selectively based on the current application scenario.

[2]. For return values -29 and -43, it is also necessary to confirm that the file path contains only English characters. If the path contains Chinese characters, the API will report a file load failure.

www.harogic.com
info@harogic.com
+65-8299 8857