V2.1.42

2026/11/20

H2 API

编程指南

HAROGIC 海得逻捷

目录

1. 版本信息 7

2. 概述 8

3. 设备及 API 版本 9

4. 函数类别简介 10

5. API 调用流程 11

5.1 CW 模式 11

5.2 实时流模式 11

5.3 回放模式 11

6. 设备与系统 device 函数 12

6.1 device_list_usb 12

6.2 device_ignore_pd 12

6.3 device_query_apiversion 12

6.4 device_open_usb 15

6.5 device_open_eth 16

6.6 device_close 18

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 20

6.13 device_config_fan 21

6.14 device_gpio_setbits 21

6.15 device_gpio_resetbits 22

6.16 device_config_power_state 22

6.17 device_query_power_state 23

6.18 device_query_temperature 23

6.19 device_config_gnss 24

6.20 device_query_gnss 24
6.21 device_query_gnss_data 24
6.22 device_config_trigger_out 25
6.23 device_query_discription_retval 28
7. 通道设置函数 29
7.1 channel_preset 29
7.2 channel_start 29
7.3 channel_power_state 29
7.4 channel_config_lo_mode 30
7.5 channel_query_lo_mode 30
7.6 channel_config_trigger 31
7.7 channel_query_trigger 31
7.8 channel_bus_trigger 32
7.9 channel_config_port 32
7.10 channel_query_port 33
7.11 channel_tx_query_capabilities 33
7.12 channel_config_amplitude_compensation 34
8. 发射设置函数 35
8.1 tx_config_output 35
8.2 tx_query_output 35
8.3 tx_config_ffm 36
8.4 tx_query_ffm 37
8.5 tx_config_fscan 38
8.6 tx_query_fscan 40
8.7 tx_config_lscan 40
8.8 tx_query_lscan 42
8.9 tx_config_mscan 43
8.10 tx_query_mscan 44
8.11 tx_clear_waveform 45

8.12 tx_download_waveform 45
8.13 tx_send_stream 51
8.14 tx_config_stream 53
8.15 tx_query_stream 54
9. 工具函数 55
9.1 device_epoch_to_utc 55
10. 调制设置函数（选件） 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 57
10.6 mod_init_pm 58
10.7 mod_init_dsss 58
10.8 mod_init_pulse 59
10.9 mod_init_multitone 59
10.10 mod_init_stepsweep 60
10.11 mod_init_rampsweep 60
10.12 mod_init_ofdm 60
10.13 mod_init_awgn 61
10.14 mod_config_digitalmod 61
10.15 mod_config_am 64
10.16 mod_config_fm 64
10.17 mod_config_pm 65
10.18 mod_config_dsss 65
10.19 mod_config_pulse 66
10.20 mod_config_multitone 66
10.21 mod_config_stepsweep 67
10.22 mod_config_rampsweep 67

10.23 mod_config_ofdm 68
10.24 mod_config_awgn 68
10.25 mod_generate_stream 69
11. 结构体变量 70
11.1 eth_setting 70
11.2 gnss_setting 70
11.3 gnss_data 70
11.4 device_info 71
11.5 option 71
11.6 channel 71
11.7 stream_trigger 72
11.8 device_trigger_out 72
11.9 gain_setting 73
11.10 agc_setting 73
11.11 stream 73
11.12 data_organization 74
11.13 processer_det 74
11.14 processer_fft 74
11.15 aux_data 75
11.16 tx_stream 76
11.17 tx_channel_capabilities 76
11.18 digitalmod_profile 77
11.19 am_profile 77
11.20 fm_profile 78
11.21 pm_profile 78
11.22 dsss_profile 78
11.23 pulse_profile 79
11.24 multitone_profile 79
11.25 stepsweep_profile 79

11.26 rampsweep_profile 79
11.27 ofdm_profile 80
11.28 awgn_profile 80
12. 枚举变量 81
12.1 mod_digitalmod_type 81
12.2 mod_filter_type 81
12.3 mod_shape_type 82
13. 宏定义 83
13.1 语言 83
13.2 选件号 83
13.3 通用状态 83
13.4 参考时钟 83
13.5 GNSS 星座 83
13.6 设备接口类型 84
13.7 通道公共定义 84
13.8 电源 84
13.9 RX 增益与 AGC 84
13.10 RX 分析器（检波方式） 85
13.11 FFT 窗型 85
13.12 TX 模式 85
13.13 触发输入源 85
13.14 触发输出源 86
13.15 触发输出脉冲极性 86
13.16 触发通道行为 86
13.17 数字调试类型 86
13.18 滤波器类型 87
13.19 调制波形 87
13.20 多音相位模式 87
附录 Appendix 1：API 返回值索引 88

1. 版本信息

版本更新说明表

版本号	内容	时间
V2.1.42	增加：SGF 系列对应软固件基线版本	2026-11-20
V2.0.40	重构：信号源 H2 API 接口 增加：宏定义章节	2026-10-14
V2.0.24	增加： device_gpio_setbits、 device_gpio_resetbits 函数	2026-7-16
V2.0.39	增加：调制设置函数（选件）章节 device_gpio_setbits	2026-5-22
V2.0.17	增加：device_query_state 函数	2026-4-20
V2.0.12	增加： 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 和 tx_query_lscan 函数	2026-4-17
V2.0.10	增加： 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 和 tx_config_stream 函数	2026-3-20
V1.0	初始版本	2026-1-10

2. 概述

此API系统是基于C编写的动态链接库，用于对设备进行编程开发。信号源主要有以下三种工作模式：

连续波模式	该模式下，设备发送连续波，用户无需向设备下发数据。
实时流模式	该模式下，用户可通过USB总线向设备发送IQ波形数据，设备接收数据并实时播放。当用户停止下发后，设备将下发的所有数据播放完成后停止播放。实时数据的采样率和传输速率受限于数据接口的物理带宽，数据大小不受限。
回放模式	该模式采用预存储波形播放机制，用户可先将IQ波形数据下载至设备存储器中，后续通过指定波形的索引、循环播放次数、采样率和播放的波形数量实现多波形播放。由于所有波形数据已预先存储，无需依赖高速实时数据传输，即可实现最大模拟带宽的波形输出。需注意的是，可存储的波形数据总量受限于设备的存储器容量。

3. 设备及 API 版本

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

版本匹配原则

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

表格 1 软固件基线版本对应表（SGA、PGA、T1000 系列）

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

表格 2 软固件基线版本对应表（SGF 系列）

API	FPGA	MCU	Bus	配置说明
2.1.42	2.0.22	2.0.36	2.0.1	F60_R0标配
	2.1.24	2.0.36	2.0.1	F60_R0 (选件53)
	2.0.22	2.0.36	2.0.1	F60_R1标配
	2.1.24	2.0.36	2.0.1	F60_R1 (选件53)

注意：

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

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

4. 函数类别简介

表格 3 API 函数类别

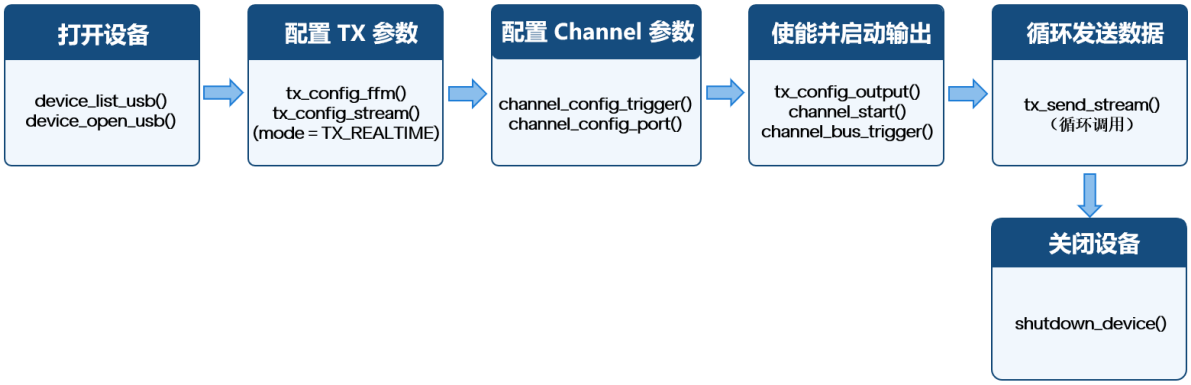
函数类别	说明
设备与系统 device	全局功能，可在任意工作模式下调用此类别下的函数。该类别包括对设备打开、关闭、全局性设置、获取设备信息和获取设备状态等功能。
通道设置 channel	通道级控制，用于管理设备内部独立的信号通路。可实现对指定通道的配置、使能、参数设置与状态查询，是进行多通道同步或独立控制的基础。
发射设置 tx	发射链路控制。此类别函数用于配置发射参数、控制信号发射流程、管理发射数据及查询发射状态，是信号源功能的核心。

5. API 调用流程

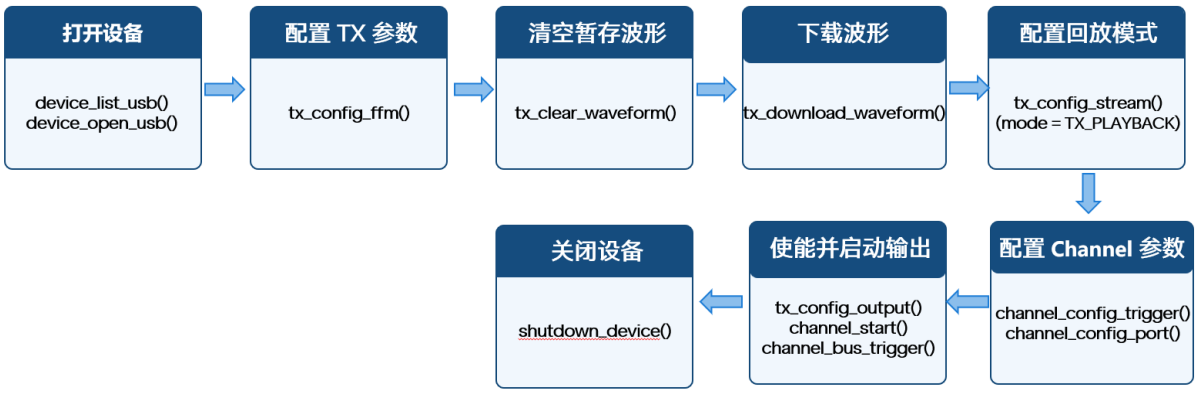
5.1 CW 模式



5.2 实时流模式



5.3 回放模式



6. 设备与系统 device 函数

在调用任何与硬件关联的 API 之前，必须首先调用 device_open_usb 初始化设备。完成业务操作后，应调用 device_close 以关闭设备并释放分配的内存资源。

6.1 device_list_usb

int device_list_usb(device_info* device_info_o, uint32_t* count)	
参数说明	
列举挂载在主机上的 USB 设备。	
兼容性	2.0.39 及之后版本支持
参数说明	
[out] device_info_o	返回 USB 设备信息数组，详见device_info结构体定义。
[out] count	返回 USB 设备总数量。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	无。
示例	请参考 device_query_apiversion 函数相关示例。

6.2 device_ignore_pd

int device_ignore_pd (uint8_t dnum)	
参数说明	
强行指定设备通过 USB 进行供电，用于忽略 USB 供电能力协商（PD），强制 USB 供电。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] dnum	USB 设备序号。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 之前调用。
示例	请参考 device_query_apiversion 函数相关示例。

6.3 device_query_apiversion

int device_query_apiversion (char version_name_o[16])
参数说明
查询API版本号。

兼容性	2.0.39 及之后版本支持
参数说明	
[out] version_name_o	返回版本字符串。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	无。
示例	

```
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
)
```

参数说明	
打开 USB 设备并获取通道信息。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **dnum**	USB 设备序号。
[in] **chs**	需要枚举的通道数。
[out] **device_o**	返回设备句柄。

[out] ch_o	返回通道数组，详见channel 结构体定义。
[out] dinfo_o	返回设备信息，详见 device_info 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	无。
示例	请参考 device_query_apiversion 函数相关示例。

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
)
```

参数说明	
打开 ETH 设备并获取通道信息。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] settings	ETH 设备启动配置，详见 eth_setting 结构体定义。
[in] chs	需要枚举的通道数。
[out] device_o	返回设备句柄。
[out] ch_o	返回通道数组，详见 channel 结构体定义。
[out] dinfo_o	返回设备信息，详见device_info 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	无。
示例	

```
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)	
参数说明	
关闭设备并释放相关资源。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	该函数仅应在程序执行结束时调用。调用后，将关闭与设备的连接并释放已分配的资源。若需重新使用该设备，必须再次调用 device_open_*，以重新建立连接并初始化硬件。
示例	请参考 device_query_apiversion 函数相关示例。

6.7 device_preset

int device_preset(void** device)	
参数说明	
设备复位（包含全通道复位）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。

6.8 device_query_options

int device_query_options(void** device, option option_o[], uint16_t* options_o)	
参数说明	
查询选件状态。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[out] option_o	选件数组，详见 option 结构体定义。

[out] options_o	生效的选件个数。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	请参考 device_query_apiversion 函数相关示例。

6.9 device_config_clock

```
int device_config_clock(
    void** device,
    uint8_t state_ext_ref,
    double fref,
    uint8_t state_ref_out
)
```

参数说明	
配置设备参考时钟。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] state_ext_ref	参考时钟源，取值：REF_INT（内部）/ REF_EXT（外部）。
[in] fref	参考时钟频率（Hz）。
[in] state_ref_out	参考时钟输出开关，取值：REF_OUT_OFF / REF_OUT_ON。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	请参考 device_query_apiversion 函数相关示例。

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
)
```

参数说明	
查询设备参考时钟配置。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。

[out] **state_ext_ref_o**	当前参考时钟源，取值：REF_INT / REF_EXT。
[out] **fref_o**	当前参考时钟频率（Hz）。
[out] **state_ref_out_o**	参考时钟输出开关状态，取值：REF_OUT_OFF / REF_OUT_ON。
[out] **is_locked_o**	时钟锁定状态：0 未锁定，1 锁定。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.11 device_query_power_supply

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

参数说明	
查询指定电源轨（USB_RAIL / RF_RAIL）的电压电流。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **rail**	电源轨，取值：USB_RAIL / RF_RAIL。
[out] **voltage_o**	返回电压（V）。
[out] **current_o**	返回电流（A）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.12 device_config_func

```
int device_config_func (
    void** device,
    uint8_t func,
    int8_t state
)
```

参数说明	
设备通用功能配置。	
兼容性	2.0.39 及之后版本支持

参数说明	
[in] **device**	设备句柄。
[in] **func**	功能号，取值：POWER_STATE。
[in] **state**	状态，取值：POWER_ON / POWER_OFF / POWER_SAVING。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.13 device_config_fan

```
int device_config_fan(
    void** device,
    float threshold
)
```

参数说明	
设定风扇配置。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **threshold**	风扇启动温度阈值：+INFINITY 关闭风扇，-INFINITY 始终开启，其他值表示风扇启动温度。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.14 device_gpio_setbits

```
int device_gpio_setbits(
    void** device,
    uint16_t bits
)
```

参数说明	
置位GPIO。	
兼容性	2.0.37 及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **bits**	GPIO IO口控制位，共16个；位为1的IO置高

返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。

6.15 device_gpio_resetbits

```
int device_gpio_resetbits(
    void** device,
    uint16_t bits
)
```

参数说明	
复位GPIO。	
兼容性	2.0.37 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] bits	GPIO IO口控制位，共16个；位为1的IO置低
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。

6.16 device_config_power_state

```
int device_config_power_state(
    void** device,
    uint8_t state
)
```

参数说明	
配置系统电源状态。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] state	电源状态，取值：POWER_ON / POWER_OFF / POWER_SAVING。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.17 device_query_power_state

int device_query_power_state(void** device, uint8_t* state_o)	
参数说明	
查询系统电源状态。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[out] state_o	返回电源状态，取值：POWER_ON / POWER_OFF / POWER_SAVING。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.18 device_query_temperature

int device_query_temperature(void** device, uint8_t domain, float* temperature_o)	
参数说明	
查询设备当前温度。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] domain	温度域。
[out] temperature_o	返回设备温度（摄氏度）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.19 device_config_gnss

int device_config_gnss(void** device, const gnss_setting* gnss)	
参数说明	
配置设备的 GNSS 参数。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] gnss	GNSS 配置参数，详见 gnss_setting 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	需在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.20 device_query_gnss

int device_query_gnss(void** device, gnss_setting* gnss_o)	
参数说明	
查询设备当前 GNSS 配置。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[out] gnss_o	返回设备 GNSS 配置，详见 gnss_setting 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.21 device_query_gnss_data

int device_query_gnss_data(void** device, gnss_data* data_o)
参数说明

查询设备当前 GNSS 信息。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[out] data_o	返回设备 GNSS 信息，详见 gnss_data 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 device_query_apiversion 函数相关示例。

6.22 device_config_trigger_out

```
int device_config_trigger_out(
    void** device,
    const device_trigger_out* trg
)
```

参数说明	
配置设备触发输出。 TX 触发输出以 device 为操作单位。source 为 CHANNEL0/CHANNEL1 时，recounter 决定 HOP/SWEEP；source 为 TRIGGER_IN_ROUTE 时，输出原始触发事件。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] trg	触发输出配置，详见 device_trigger_out 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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;
}
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)	
参数说明	
查询返回值描述与建议。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] retval	函数返回值（见上方 H2_ERROR_* / H2_WARNING_* 定义）。
[in] language	语言参数，取值：H2_LANGUAGE_ENGLISH / H2_LANGUAGE_CHINESE（保留字段，当前忽略，始终输出英文）。
[out] discription	描述字符串，不超过 128 字节（含终止符）。
[out] suggestion	建议字符串，不超过 128 字节（含终止符）。
返回值	H2_NOERROR 成功，H2_WARNING_PARAMOUTRANGE 参数无效。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。

7. 通道设置函数

7.1 channel_preset

int channel_preset(const channel* ch)	
参数说明	
对通道执行复位操作。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **ch**	目标通道，详见 channel 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.2 channel_start

int channel_start(const channel* ch)	
参数说明	
启动通道运行。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **ch**	目标通道，详见 channel 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.3 channel_power_state

int channel_power_state (const channel* ch, uint8_t state)	
参数说明	
设置通道电源状态。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] **ch**	目标通道，详见 channel 结构体定义。

[in] state	电源状态，取值：POWER_ON / POWER_OFF / POWER_SAVING。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.4 channel_config_lo_mode

```
int channel_config_lo_mode (
    channel* ch,
    uint8_t mode
)
```

参数说明	
配置通道本振模式。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] mode	本振模式，取值：LOMODE_AUTO / LOMODE_HIGHSPEED / LOMODE_LOWSPUR / LOMODE_LOWPHASENOISE。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.5 channel_query_lo_mode

```
int channel_query_lo_mode (
    channel* ch,
    uint8_t* mode_o
)
```

参数说明	
查询通道本振模式。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] mode_o	返回本振模式，取值：LOMODE_AUTO / LOMODE_HIGHSPEED / LOMODE_LOWSPUR / LOMODE_LOWPHASENOISE。

返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.6 channel_config_trigger

```
int channel_config_trigger(
    channel* ch,
    int16_t strm,
    uint8_t ch_action
)
```

参数说明	
指定 stream 触发事件到来后的通道响应行为。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] strm	stream 序号；当前固定使用 stream0。
[in] ch_action	通道行为，TRIGGER_ACTION_HOP / TRIGGER_ACTION_SWEEP。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.7 channel_query_trigger

```
int channel_query_trigger(
    channel* ch,
    int16_t* strm_o,
    uint8_t* ch_action_o
)
```

参数说明	
查询通道触发响应配置。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] strm_o	返回实际绑定的 stream 序号（当前为 0）。
[out] ch_action_o	返回通道行为；动作次数由 tx_query_stream() 返回的 stream_trigger::response_count 指定。

返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.8 channel_bus_trigger

```
int channel_bus_trigger(
    const channel* ch,
    uint32_t stream_mask
)
```

参数说明	
配置总线触发。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] stream_mask	stream（stream序号掩码，当前未生效，仅能配置成0）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.9 channel_config_port

```
int channel_config_port (
    channel* ch,
    uint16_t port
)
```

参数说明	
设置射频输出端口。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] port	端口号：0 标准输出端口，1 高功率端口。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

7.10 channel_query_port

int channel_query_port (channel* ch, uint16_t* port_o)	
参数说明	
查询射频输出端口。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] port_o	返回端口号：0 标准输出端口，1 高功率端口。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。

7.11 channel_tx_query_capabilities

int channel_tx_query_capabilities(channel* ch, tx_channel_capabilities* capabilities)	
参数说明	
查询 TX 通道能力。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] capabilities	返回 TX 通道能力，详见 tx_channel_capabilities 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。

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)	
参数说明	
配置幅度补偿。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] enable	使能：0 关闭补偿，1 开启补偿。
[in] freq	频率数组。
[in] amplitude	幅度补偿值数组。
[in] points	频响补偿数据点数。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。

8. 发射设置函数

8.1 tx_config_output

int tx_config_output (channel* ch, uint8_t state_rfout, uint8_t state_mod)	
功能描述	
配置通道的输出状态（RFOUT / MOD）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] state_rfout	射频输出使能，取值：STATE_OFF / STATE_ON。
[in] state_mod	调制输出使能，取值：STATE_OFF / STATE_ON。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

8.2 tx_query_output

int tx_query_output(channel* ch, uint8_t* rfout_o, uint8_t* mod_o)	
功能描述	
查询通道的输出状态（RFOUT / MOD）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] rfout_o	返回射频输出状态，取值：STATE_OFF / STATE_ON。
[out] mod_o	返回调制输出状态，取值：STATE_OFF / STATE_ON。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

8.3 tx_config_ffm

int tx_config_ffm(channel* ch, double fc, float level)	
功能描述	
配置通道的频率和幅度参数（FFM）。	
兼容性	1.0.1 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] fc	中心频率（Hz）。
[in] level	输出幅度（dBm）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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
)
```

功能描述	
查询通道的频率和幅度参数（FFM）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] fc_o	返回中心频率（Hz）。
[out] level_o	返回输出幅度（dBm）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

8.5 tx_config_fscan

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

功能描述	
配置通道的频率扫描参数（FSCAN）。在固定功率 level 下，从 start 开始，到 stop 为止，以 step 为步进递增频率，每个频点驻留 dwell。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] start	扫描起始频率（Hz）。
[in] stop	扫描终止频率（Hz）。
[in] step	扫描步进频率（Hz）。
[in] level	输出幅度（dBm）。
[in] dwell	单个频点驻留时间（s）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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)	
功能描述	
查询通道的频率扫描参数（FSCAN）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] start_o	返回扫描起始频率（Hz）。
[out] stop_o	返回扫描终止频率（Hz）。
[out] step_o	返回扫描步进频率（Hz）。
[out] level_o	返回输出幅度（dBm）。
[out] dwell_o	返回单个频点驻留时间（s）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_fscan 函数相关示例。

8.7 tx_config_lscan

int tx_config_lscan(channel* ch, double fc, float level_start, float level_stop, float level_step, float dwell)	
功能描述	
配置通道的幅度扫描参数（LSCAN）。在固定中心频率 fc 下，从 level_start 开始，到 level_stop 为止，以 level_step 为步进递增幅度，每个幅度点驻留 dwell。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。

[in] fc	中心频率（Hz）。
[in] level_start	扫描起始幅度（dBm）。
[in] level_stop	扫描终止幅度（dBm）。
[in] level_step	扫描步进幅度（dB）。
[in] dwell	单个幅度点驻留时间（s）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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
)
```

功能描述	
查询通道的幅度扫描参数（LSCAN）。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] fc_o	返回中心频率（Hz）。
[out] level_start_o	返回扫描起始幅度（dBm）。
[out] level_stop_o	返回扫描终止幅度（dBm）。
[out] level_step_o	返回扫描步进幅度（dB）。
[out] dwell_o	返回单个幅度点驻留时间（s）。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_lscan 函数相关示例。

8.9 tx_config_mscan

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

功能描述	
配置通道的多点扫描参数（MSCAN）。按数组下标顺序，从第 0 点到第 n-1 点依次扫描，每个点使用对应的 fc[i] 与 level[i]。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] fc[]	频率数组，长度为 n。
[in] level[]	幅度数组，长度为 n。
[in] dwell[]	驻留时间数组（s），长度为 n。
[in] n	扫描点个数。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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
)
```

功能描述

查询通道的多点扫描参数（MSCAN）。按数组下标顺序返回当前配置的扫描序列。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[out] fc_o	返回频率数组，长度为 n。
[out] level_o	返回幅度数组，长度为 n。
[out] dwell_o	返回驻留时间数组（s），长度为 n。
[out] n_o	返回扫描点个数。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_mscan 函数相关示例。

8.11 tx_clear_waveform

int tx_clear_waveform(void device)**	
功能描述	
清除设备中所有波形数据。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_download_waveform 函数相关示例。

8.12 tx_download_waveform

```
int tx_download_waveform(
    void** device,
    int16_t iq[],
    uint32_t points,
    uint16_t* waveform_o
)
```

功能描述	
向设备下载波形数据。仅支持波形按照 waveform = 0, 1, 2, 3... 的顺序下载。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] device	设备句柄。
[in] iq	IQ 数据。

[in] points	IQ 点数。
[out] waveform_o	返回波形序号。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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)	
功能描述	
向指定通道发送 IQ 数据进行实时播放。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] strm	stream 序号（当前仅 strm0 可用）。
[in] iq	IQ 数据。
[in] points	IQ 点数。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	

```
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
)
```

功能描述	
配置发射 stream。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	通道句柄。
[in] strm	stream序号；当前非 0 自动归一化为 0。
[in] setting	tx stream 业务配置，详见 tx_stream 结构体定义。
[in] trg	stream 触发配置；成功后由 Host 缓存，详见 stream_trigger 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

8.15 tx_query_stream

int tx_query_stream(const channel* ch, int16_t strm, tx_stream* setting_o, stream_trigger* trg_o)	
功能描述	
查询发射 stream 配置。	
兼容性	2.0.39 及之后版本支持
参数说明	
[in] ch	目标通道，详见 channel 结构体定义。
[in] strm	stream序号；当前非 0 自动归一化为 0。
[out] setting_o	返回 tx stream 业务配置，详见 tx_stream 结构体定义。
[out] trg_o	返回 Host 缓存的 stream 触发配置；fmt 指针在下次配置、预置或关闭设备前有效。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_config_ffm 函数相关示例。

9. 工具函数

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
)
```

功能描述	
将 unix Epoch 纳秒时间戳转换为可读的 UTC 时间。	
兼容性	2.0.39 及之后版本支持
参数说明	
[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.
返回值	无。
调用约束	在 device_open_usb 或者 device_open_eth 函数后调用。
示例	参考 tx_send_stream 函数相关示例。

10. 调制设置函数（选件）

10.1 mod_open

int mod_open(void** device, char version_o[50])	
功能描述	
开启调制功能，设备打开后，将设备信息结构体传入函数用于校验设备，检查调制库是否存在，注册函数并创建设备对象，打开调制功能。	
兼容性	2.0.39及之后版本支持
参数说明	
[out] **device**	返回设备句柄。在调用后续API时，使用该句柄引用目标设备。
[out] **version_o[50]**	返回调制库版本号
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行device_open_*初始化之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。

10.2 mod_close

int mod_close(void** device)	
功能描述	
关闭调制功能	
兼容性	2.0.39及之后版本支持
参数说明	
[out] **device**	设备句柄。关闭后该句柄将失效。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	仅在调制程序执行结束前调用此函数。执行后，调制功能将关闭且相关资源被释放。如需再次开启调制功能，必须重新调用mod_open。
示例	请参考 mod_config_digitalmod 函数相关示例。

10.3 mod_init_digitalmod

int mod_init_digitalmod(void** device, digitalmod_profile* digitalmod_profile_o)	
功能描述	
初始化数字调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **digitalmod_profile_o**	数字调制配置结构体。详见digitalmod_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。

10.4 mod_init_am

int mod_init_am(void** device, am_profile* am_profile_o)	
功能描述	
初始化AM调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **am_profile_o**	AM调制配置结构体。详见 am_profile 结构体定义。
调用约束	必须在成功执行 mod_open 之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将 profile/init/config 函数替换为对应调制类型的函数。

10.5 mod_init_fm

int mod_init_fm(void** device, fm_profile* fm_profile_o)
功能描述
初始化FM调制配置结构体，将结构体中的每个参数都赋初值。

兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **fm_profile_o**	FM调制配置结构体。详见fm_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.6 mod_init_pm

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

功能描述	
初始化PM调制配置结构体，默认值为1 kHz正弦调制、1 rad相位偏移、0 rad初始相位。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **pm_profile_o**	PM调制配置结构体。详见pm_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.7 mod_init_dsss

```
int mod_init_dsss(
    void** device,
    dsss_profile* dsss_profile_o
)
```

功能描述	
初始化直接序列扩频配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **dsss_profile_o**	直接序列扩频配置结构体。详见dsss_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。

示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.8 mod_init_pulse

```
int mod_init_pulse(
    void** device,
    pulse_profile* pulse_profile_o
)
```

功能描述	
初始化脉冲调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **pulse_profile_o**	脉冲调制配置结构体。详见pulse_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.9 mod_init_multitone

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

功能描述	
初始化多音配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **multitone_profile_o**	多音配置结构体。详见multitone_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.10 mod_init_stepsweep

int mod_init_stepsweep(void** device, stepsweep_profile* stepsweep_profile_o)	
功能描述	
初始化步进扫描配置结构体，将结构体中的每个参数都赋初值	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **stepsweep_profile_o**	步进扫描配置结构体。详见stepsweep_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.11 mod_init_rampsweep

int mod_init_rampsweep(void** device, rampsweep_profile* rampsweep_profile_o)	
功能描述	
初始化斜坡扫描配置结构体，将结构体中的每个参数都赋初值	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **rampsweep_profile_o**	斜坡扫描配置结构体。详见rampsweep_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.12 mod_init_ofdm

int mod_init_ofdm (void** device, ofdm_profile* ofdm_profile_o)
功能描述

初始化正交频分复用配置结构体，将结构体中的每个参数都赋初值	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **ofdm_profile_o**	正交频分复用配置结构体。详见ofdm_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.13 mod_init_awgn

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

功能描述	
初始化高斯白噪声配置结构体，将结构体中的每个参数都赋初值	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[out] **awgn_profile_o**	正交频分复用配置结构体。详见awgn_profile结构体定义。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.14 mod_config_digitalmod

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

功能描述	
初始化数字调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。

[in] **digitalmod_profile_in**	输入数字调制配置结构体。详见digitalmod_profile结构体定义。
[out] **digitalmod_profile_o**	输出数字调制配置结构体。详见digitalmod_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_open之后调用此函数。
示例	

```
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
)
```

功能描述	
初始化AM调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **am_profile_in**	输入AM调制配置结构体。详见am_profile结构体定义。
[out] **am_profile_o**	输出AM调制配置结构体。详见am_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_am之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.16 mod_config_fm

```
int mod_config_fm(
    void** device,
    const fm_profile* fm_profile_in
    fm_profile* fm_profile_o
)
```

功能描述	
配置配置FM调制参数，若输入配置不合理，将回写输出配置结构体为合理参数	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **fm_profile_in**	输入FM调制配置结构体。详见fm_profile结构体定义。

[out] **fm_profile_o**	输出FM调制配置结构体。详见fm_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_fm之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.17 mod_config_pm

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

功能描述	
配置配置pM调制参数，若输入配置不合理，将回写输出配置结构体为合理参数	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **pm_profile_in**	输入PM调制配置结构体。详见pm_profile结构体定义。
[out] **pm_profile_o**	输出PM调制配置结构体。详见pm_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_fm之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.18 mod_config_dsss

```
int mod_config_dsss(
    void** device,
    const dsss_profile* dsss_profile_in,
    dsss_profile* dsss_profile_o
)
```

功能描述	
配置直接序列扩频参数，若输入配置不合理，将回写输出配置结构体为合理参数。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **dsss_profile_in**	输入直接序列扩频配置结构体。详见dsss_profile结构体定义。

[out] **dsss_profile_o**	输出直接序列扩频配置结构体。详见dsss_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_dsss之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.19 mod_config_pulse

```
int mod_config_pulse(
    void** device,
    const pulse_profile* pulse_profile_in
    pulse_profile* pulse_profile_o
)
```

功能描述	
初始化脉冲调制配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **pulse_profile_in**	输入脉冲调制配置结构体。详见pulse_profile结构体定义。
[out] **pulse_profile_o**	输出脉冲调制配置结构体。详见pulse_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_pulse之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.20 mod_config_multitone

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

功能描述	
初始化多音配置结构体，将结构体中的每个参数都赋初值。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **multitone_profile_in**	输入多音配置结构体。详见multitone_profile结构体定义。

[out] **multitone_profile_o**	输出多音配置结构体。详见multitone_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_multitone之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.21 mod_config_stepsweep

```
int mod_config_stepsweep(
    void** device,
    const stepsweep_profile* stepsweep_profile_in
    stepsweep_profile* stepsweep_profile_o
)
```

功能描述	
配置步进扫描参数，若输入配置不合理，将回写输出配置结构体为合理参数	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **stepsweep_profile_in**	输入步进扫描配置结构体。详见stepsweep_profile结构体定义。
[out] **stepsweep_profile_o**	输入步进扫描配置结构体。详见stepsweep_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_stepsweep之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

10.22 mod_config_rampsweep

```
int mod_config_rampsweep(
    void** device,
    const rampsweep_profile* rampsweep_profile_in
    rampsweep_profile* rampsweep_profile_o
)
```

功能描述	
配置斜坡扫描参数，若输入配置不合理，将回写输出配置结构体为合理参数	
兼容性	2.0.39及之后版本支持
参数说明	

[in] **device**	设备句柄。
[out] **rampsweep_profile_o**	斜坡扫描配置结构体。详见 rampsweep_profile 结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行 mod_init_rampsweep 之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将 profile/init/config 函数替换为对应调制类型的函数。

10.23 mod_config_ofdm

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

功能描述	
配置正交频分复用参数，若输入配置不合理，将回写输出配置结构体为合理参数。	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **ofdm_profile_o**	输入正交频分复用配置结构体。详见ofdm_profile结构体定义。
[out] **ofdm_profile_o**	输出正交频分复用配置结构体。详见ofdm_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_ofdm之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将 profile/init/config 函数替换为对应调制类型的函数。

10.24 mod_config_awgn

```
int mod_config_awgn (
    void** device,
    const awgn_profile* awgn_profile_in
    awgn_profile* awgn_profile_o
)
```

功能描述	
初始化高斯白噪声配置结构体，将结构体中的每个参数都赋初值	
兼容性	2.0.39及之后版本支持

参数说明	
[in] **device**	设备句柄。
[in] **awgn_profile_in**	输入高斯白噪声配置结构体。详见awgn_profile结构体定义。
[out] **awgn_profile_o**	输出高斯白噪声配置结构体。详见awgn_profile结构体定义。
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在成功执行mod_init_awgn之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

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
)
```

功能描述	
打开调制功能后，调用初始化函数赋初值，然后通过配置结构体函数下发参数，生成信号数据	
兼容性	2.0.39及之后版本支持
参数说明	
[in] **device**	设备句柄。
[in] **symbol_length**	符号长度，用于数字调制、直接序列扩频调制。
[out] **iq_o**	输出信号数据
[out] **length_o**	输出数据长度
[out] **sample_rate_o**	当前生成数据所用的采样率
返回值	0：成功；非0：错误代码，详见附录1。
调用约束	必须在执行 device_open_* 初始化之后调用此函数。
示例	请参考 mod_config_digitalmod 函数相关示例。将profile/init/config 函数替换为对应调制类型的函数。

11. 结构体变量

11.1 eth_setting

eth_setting 详细定义	
uint8_t is_ipv6	协议类型：0 IPv4，1 IPv6
uint8_t ip_address[16]	设备 IP 地址（IPv4 用前 4 字节）
uint16_t remote_port	远端端口
int32_t read_timeout	读取超时时间（ms）
void* extension1	预留扩展
void* extension2	预留扩展

11.2 gnss_setting

gnss_setting 详细定义	
uint8_t antenna	天线选择：0 内部天线，1 外部天线
int8_t constellations	星座使能，取值：CONSTELLATION_GPS / CONSTELLATION_GLONASS / CONSTELLATION_BEIDOU / CONSTELLATION_GALILEO，按位或
uint8_t state_pps_out	PPS 输出使能，取值：STATE_OFF / STATE_ON
float pps_x	PPS 频率（Hz）
float pps_delay	PPS 延时（s）
void* extension1	预留扩展
void* extension2	预留扩展

11.3 gnss_data

gnss_data 详细定义	
uint8_t is_locked	GNSS 锁定状态：0 未锁定，1 锁定
uint8_t sat_nums	当前定位卫星数量
uint8_t snr_max	卫星 SNR 最大值
uint8_t snr_avg	卫星 SNR 平均值
uint8_t snr_min	卫星 SNR 最小值
float latitude	纬度，北纬为正，南纬为负
float longitude	经度，东经为正，西经为负

float altitude	海拔（m）
uint64_t ns_sinceepoch	从 Epoch 起累计的时间（ns）
void* extension1	预留扩展
void* extension2	预留扩展

11.4 device_info

device_info 详细定义	
uint16_t model	设备型号
char model_name[16]	设备型号字符串
uint8_t chn	channel 个数
uint64_t uid_l64	低 64bit 设备序列号
uint32_t uid_h32	高 32bit 设备序列号
uint8_t versions	版本数量
char version_name[16][16]	版本字符串数组
uint8_t interfaces	接口数量
char interface_name[8][16]	接口名称数组
void* extension1	预留扩展
void* extension2	预留扩展

11.5 option

option 详细定义	
uint16_t code	选件号，取值：OPTION_MEDIUM_POWER / OPTION_PULSE_DET / OPTION_EMI / OPTION_BW_100M_RX / OPTION_BW_320M_TX
char description[64]	选件描述
uint8_t state	选件状态：0 未启用，1 启用
uint32_t data[16]	选件附加数据
void* extension1	预留扩展
void* extension2	预留扩展

11.6 channel

channel 详细定义	
void** device	通道所属设备句柄

uint16_t num	通道号
uint8_t type	通道类型，取值：CHANNELTYPE_RX / CHANNELTYPE_TX
uint16_t max_streams	通道支持的 stream 数量上限
void* extension1	预留扩展
void* extension2	预留扩展

11.7 stream_trigger

stream_trigger 详细定义	
uint8_t source	触发源
int64_t response_count	响应触发的次数；-1 表示无限次响应，>= 0 表示响应有限个触发后停止响应
uint64_t triggerdelay	触发延迟点数（暂不支持）
uint64_t pretrigger	预触发点数（暂不支持）
uint64_t retrigger_interval	重触发间隔点数（暂不支持）
uint64_t retrigger_count	重触发次数；设备自激，不计入 response_count（暂不支持）
float trigger_level	触发门限（暂不支持）
uint64_t safetime	安全点数（暂不支持）
float* fmt	MASK 数组（暂不支持）
uint64_t fmt_points	模板点数（暂不支持）
void* extension1	预留扩展
void* extension2	预留扩展

11.8 device_trigger_out

device_trigger_out 详细定义	
uint8_t state	触发输出使能，取值：STATE_OFF / STATE_ON
uint8_t source	TX 触发输出来源，取值：CHANNEL0 / CHANNEL1 / TRIGGER_IN_ROUTE
uint8_t negative_pulse	输出脉冲极性，TRIGGER_PULSE_POSITIVE / TRIGGER_PULSE_NEGATIVE
uint32_t recounter	通道源时：1 输出 HOP，>1 输出 SWEEP；EVENT 源时忽略
void* extension1	预留扩展
void* extension2	预留扩展

11.9 gain_setting

gain_setting 详细定义	
float ref_level	参考电平（dBm）
uint8_t state_manual_atten	手动衰减使能，取值：STATE_OFF / STATE_ON
uint8_t atten	手动衰减值（dB）
uint8_t amplifers	增益配置，取值：AMPLIFERS_AUTO / AMPLIFERS_PREAMP_ON_PATHGAIN_HIGH / AMPLIFERS_PREAMP_OFF_PATHGAIN_HIGH / AMPLIFERS_PREAMP_OFF_PATHGAIN_MEDIUM / AMPLIFERS_PREAMP_OFF_PATHGAIN_LOW
void* extension1	预留扩展
void* extension2	预留扩展

11.10 agc_setting

agc_setting 详细定义	
uint8_t state	AGC 使能，取值：STATE_OFF / STATE_ON
uint8_t mode	AGC 模式，取值：AGC_MODE_AUTO
float target_level	AGC 目标电平（dBm）
void* extension1	预留扩展
void* extension2	预留扩展

11.11 stream

stream 详细定义	
uint8_t state	stream 使能，取值：STATE_OFF / STATE_ON
uint8_t state_output	数据输出使能，取值：STATE_OFF / STATE_ON
double samplerate	采样率（Hz）
double fc_offset	中心频率偏移（Hz）
float phase_offset	相位偏移（rad）
float scale	幅度缩放
uint8_t state_32bit_iq	32bit IQ 输出使能，取值：STATE_OFF / STATE_ON
void* extension1	预留扩展

void* extension2	预留扩展

11.12 data_organization

data_organization 详细定义	
uint8_t point_size	单点字节数
uint32_t hops_in_sweep	一次扫描（sweep）中的跳频（hop）数
uint32_t packets_in_hop	一个 hop 中的数据包（packet）数
uint32_t frames_in_packet	一个 packet 中的帧（frame）数
uint32_t points_in_frame	一帧中的点数
void* extension1	预留扩展
void* extension2	预留扩展

11.13 processer_det

processer_det 详细定义	
uint8_t state	检波处理器使能，取值：STATE_OFF / STATE_ON
uint8_t state_output	数据输出使能，取值：STATE_OFF / STATE_ON
double rbw	分辨率带宽（Hz）
uint8_t state_6db_rbw	6dB RBW 使能，取值：STATE_OFF / STATE_ON
uint8_t trace_detector	迹线检波方式，取值：DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS
float trace_detector_ratio	迹线检波比例
void* extension1	预留扩展
void* extension2	预留扩展

11.14 processer_fft

processer_fft 详细定义	
uint8_t state	FFT 处理器使能，取值：STATE_OFF / STATE_ON
uint8_t state_output_frame	帧数据输出使能，取值：STATE_OFF / STATE_ON
uint8_t state_output_psacn	扫描（pscan）数据输出使能，取值：STATE_OFF / STATE_ON
uint8_t state_output_bitmap	位图数据输出使能，取值：STATE_OFF / STATE_ON

double rbw	分辨率带宽（Hz）
double vbw	视频带宽（Hz）
uint8_t window	FFT 窗型，取值：FFTWINDOW_NUTTALL / FFTWINDOW_FLATTOP / FFTWINDOW_LOWSIDELOBE / FFTWINDOW_GUASSIAN
float overlap	帧间重叠率
uint8_t frame_detector	帧检波方式，取值：DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS
float frame_detector_ratio	帧检波比例
uint8_t trace_detector	迹线检波方式，取值：DETECTOR_MAX / DETECTOR_MIN / DETECTOR_AVG / DETECTOR_RMS / DETECTOR_Samp_oLE / DETECTOR_QP / DETECTOR_MAXPOWER / DETECTOR_AUTOPEAK / DETECTOR_BYPASS
float trace_detector_ratio	迹线检波比例
uint8_t state_8bit_amplitude	8bit 幅度输出使能，取值：STATE_OFF / STATE_ON
void* extension1	预留扩展
void* extension2	预留扩展

11.15 aux_data

aux_data 详细定义	
double fc	当前中心频率（Hz）
double bin	当前频率分辨率（Hz）
float scale	幅度缩放
float offset	幅度偏移
uint32_t hops_in_sweep	一次扫描中的跳频数
uint32_t packets_in_hop	一个 hop 中的数据包数
uint32_t frames_in_packet	一个 packet 中的帧数
uint32_t points_in_frame	一帧中的点数
uint32_t sweepi	当前扫描序号
uint32_t hopi	当前跳频序号
uint32_t packeti	当前数据包序号
int32_t underrun	数据欠载计数

float max_power	最大功率
uint32_t max_power_index	最大功率对应的索引
float temperature	设备温度
int32_t health	设备健康状态
gnss_data gnss_data	GNSS 信息，详见 gnss_data 结构体定义。
void* extension1	预留扩展
void* extension2	预留扩展

11.16 tx_stream

tx_stream 详细定义	
uint8_t state	使能，取值：STATE_OFF / STATE_ON
uint8_t mode	工作模式，取值：TX_CW / TX_REALTIME / TX_PLAYBACK
double fc_offset	频率偏移（Hz）
float phase_offset	相位偏移（rad）
double realtime_srate	实时播放：采样率（Hz）
int16_t waveforms	回放：波形个数
int16_t* waveform	回放：waveform[i]，波形在内存中的索引
int32_t* repeat	回放：repeat[i]，波形的回放次数
double* playback_srate	回放：srate[i]：波形的采样率
void* extension1	预留扩展
void* extension2	预留扩展

11.17 tx_channel_capabilities

tx_channel_capabilities 详细定义	
double freq_min	最小中心频率（Hz）
double freq_max	最大中心频率（Hz）
double freq_step_min	最小频率步进（Hz）
double freq_step_max	最大频率步进（Hz）
double bandwidth_min	最小带宽（Hz）
double bandwidth_max	最大带宽（Hz）
double level_min	最小输出幅度（dBm）
double level_max	最大输出幅度（dBm）
double level_step_min	最小幅度步进（dB）

double level_step_max	最大幅度步进（dB）
double dwelltime_min	最小驻留时间（s）
double dwelltime_max	最大驻留时间（s）
double realtime_srate_min	实时播放最小采样率（Hz）
double realtime_srate_max	实时播放最大采样率（Hz）
double playback_srate_min	回放最小采样率（Hz）
double playback_srate_max	回放最大采样率（Hz）
uint64_t memory_size	波形存储容量（Byte）
void* extension1	预留扩展
void* extension2	预留扩展

11.18 digitalmod_profile

digitalmod_profile 详细定义	
uint8_t digitalmod_type	数字调制方式，取值参考数字调制类型宏定义。
double symbolrate	符号速率，单位 Hz， 范围：最小采样率/每个符号样点数 ~ 最大采样率/每个符号样点数
double fskdeviation	FSK 频偏，单位 Hz,最大为 15 * symbolrate
int32_t pn	伪随机序列，4 ~ 32，建议最大不超过 24
uint8_t filter_type	滤波器类型，取值: MOD_FILTER_RECTANGULAR / MOD_FILTER_RAISED_COSINE / MOD_FILTER_ROOT_RAISED_COSINE / MOD_FILTER_GAUSSIAN / MOD_FILTER_HALF_SINE
int32_t filter_symbols	滤波器符号数，偶数，2 ~ sps*filter_symbols <= 400, 默认取 8
int32_t sps	每个符号样点数，4 ~ 32 的偶数，默认取 4
double alpha	升余弦/根升余弦：0.025 ~ 1，高斯滤波：0.15 ~ 2.5，alpha 只对升余弦、根升余弦、高斯滤波有效

11.19 am_profile

am_profile 详细定义	
double rate	调制率，单位 Hz
double depth	调制深度，单位%
uint8_t shape	调制波形，取值：MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.20 fm_profile

fm_profile 详细定义	
double rate	调制率，单位 Hz
double deviation	调制频偏，单位 Hz
uint8_t shape	调制波形，取值：MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.21 pm_profile

pm_profile 详细定义	
double rate	调制率，单位 Hz
double phase_deviation	最大相位偏移，单位 rad，范围[0, 2*pi]
double init_phase	初始相位，单位 rad，配置是归一化到[-pi, pi]
uint8_t shape	调制波形，取值：MOD_SHAPE_SINE / MOD_SHAPE_SQUARE / MOD_SHAPE_TRIANGLE / MOD_SHAPE_RAMP

11.22 dsss_profile

dsss_profile 详细定义	
double symbolrate	符号速率，单位 Hz
uint8_t digitalmod_type	数字调制方式，取值参考数字调制类型宏定义。
int32_t code	扩频码，取 PN 值，建议取 5
int32_t filter_symbols	滤波器符号数，偶数，2 ~ sps*filter_symbols <= 400, 默认取 8
int32_t sps	每个符号样点数，4 ~ 32 的偶数，默认取 4
double alpha	升余弦/根升余弦：0.025 ~ 1，高斯滤波：0.15 ~ 2.5，alpha 只对升余弦、根升余弦、高斯滤波有效
int32_t sequenceseed	伪随机序列初始值，1 ~ 2147483647
uint8_t filter_type	滤波器类型，取值：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 详细定义	
double width	脉宽，单位 s
double period	周期，单位 s，周期一定大于脉宽

11.24 multitone_profile

multitone_profile 详细定义	
uint8_t phase_mode	相位模式，取值：MOD_MULTITONE_PHASE_FIXED / MOD_MULTITONE_PHASE_RANDOM / MOD_MULTITONE_PHASE_PARABOLIC
uint32_t seed	MOD_MULTITONE_PHASE_RANDOM 模式随机种子；其他模式忽略
int32_t tonecount	候选单音数量，范围[2, MOD_MULTITONE_MAX_TONES]
double freqsapcing	相邻音频率间隔，单位 Hz
int32_t generate_tones	要生成的候选 tone 数，等于 tonecount 表示全部输出
const int32_t* generate_tone_idx	要生成的候选 tone 下标，范围[0, tonecount-1]；全播时可为 nullptr

11.25 stepsweep_profile

stepsweep_profile 详细定义	
double center	中心频率，单位 Hz
double span	扫宽，单位 Hz
int32_t points	点数
double dwelltime	驻留时间，单位 s

11.26 rampsweep_profile

rampsweep_profile 详细定义	
double span	扫宽，单位 Hz
double sweeptime	扫描时间，单位 s
double period	周期，单位 s

11.27 ofdm_profile

ofdm_profile 详细定义	
double idleinterval	发射突发之间的空闲时间，单位 s
int32_t fftsize	FFT 个数，支持 16、32、64、128、256、512、1024、2048， 且 fftsize > guardbandcarriers_left + guardbandcarriers_right
int32_t symbolcount	符号数量
int32_t guardbandcarriers_left	左侧保护载波个数
int32_t guardbandcarriers_right	右侧保护载波个数
int32_t guardinterval	保护间隔，单位为%
int32_t window_length	窗长度，单位为%
uint8_t windowed	是否加窗
uint8_t digitalmod_type	数字调制方式，取值参考数字调制类型宏定义。
uint8_t nulldc	直流是否置空
double samplerate	采样率 195.3125e3 ~ 125e6，默认 20MHz

11.28 awgn_profile

awgn_profile 详细定义	
double bandwidth	发射突发之间的空闲时间，单位 s
int32_t length	FFT 个数，支持 16、32、64、128、256、512、1024、2048， 且 fftsize > guardbandcarriers_left + guardbandcarriers_right

12. 枚举变量

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	矩形滤波器
FilterType_RaisedCosine	升余弦滤波器
FilterType_RootRaisedCosine	根升余弦滤波器
FilterType_Gaussian	高斯滤波器
FilterType_HalfSine	半正弦滤波器

12.3 mod_shape_type

ModShape_Sine	正弦波
ModShape_Square	方波
ModShape_Triangle	三角波
ModShape_Ramp	锯齿波

13. 宏定义

13.1 语言

H2_LANGUAGE_ENGLISH	0	英语
H2_LANGUAGE_CHINESE	1	中文

13.2 选件号

OPTION_MEDIUM_POWER	10	中功率选件
OPTION_PULSE_DET	72	脉冲解调选件
OPTION_EMI	73	EMI 预兼容测量选件
OPTION_BW_100M_RX	50	100 MHz 分析带宽选件
OPTION_BW_320M_TX	53	320 MHz 信号源带宽选件

13.3 通用状态

AUTO	(-1)	自动
OFF	0	关闭
ON	1	开启
STATE_OFF	0	状态：关闭
STATE_ON	1	状态：开启

13.4 参考时钟

REF_INT	0	参考时钟源：内部
REF_EXT	1	参考时钟源：外部
REF_OUT_OFF	0	参考时钟输出：关闭
REF_OUT_ON	1	参考时钟输出：开启
INTERNAL	0	内部
EXTERNAL	1	外部

13.5 GNSS 星座

CONSTELLATION_GPS	1	GPS 星座（bit0）
CONSTELLATION_GLONASS	2	GLONASS 星座（bit1）
CONSTELLATION_BEIDOU	4	北斗星座（bit2）
CONSTELLATION_GALILEO	8	GALILEO 星座（bit3）

13.6 设备接口类型

INTERFACE_USB	0	USB 接口
INTERFACE_ETH_1000M	1	千兆以太网接口
INTERFACE_ETH_10G	2	10G 以太网接口
INTERFACE_ETH_25G	3	25G 以太网接口
INTERFACE_ETH_40G	4	40G 以太网接口
INTERFACE_ETH_100G	5	100G 以太网接口

13.7 通道公共定义

ALLCHANNEL	-1	通道全选
MAXCHANNELS	8	通道个数上限
CHANNELTYPE_RX	0	通道类型：接收通道
CHANNELTYPE_TX	1	通道类型：发射通道
CHANNEL0	0	通道序号 0
CHANNEL1	1	通道序号 1
LOMODE_AUTO	0	本振模式：自动
LOMODE_HIGHSPEED	1	本振模式：高速
LOMODE_LOWSPUR	2	本振模式：低杂散
LOMODE_LOWPHASENOISE	3	本振模式：低相位噪声

13.8 电源

USB_RAIL	0	电源轨：USB 端口
RF_RAIL	1	电源轨：RF 端口
POWER_STATE	1	func 取值：电源状态
POWER_ON	0	电源状态：开机
POWER_OFF	1	电源状态：关机
POWER_SAVING	2	电源状态：省电

13.9 RX 增益与 AGC

AMPLIFERS_AUTO	0	增益：自动
AMPLIFERS_PREAMP_ON_PATHGAIN_HIGH	1	增益：前置放大开，通路增益高
AMPLIFERS_PREAMP_OFF_PATHGAIN_HIGH	2	增益：前置放大关，通路增益高
AMPLIFERS_PREAMP_OFF_PATHGAIN_MEDIUM	3	增益：前置放大关，通路增益中

AMPLIFERS_PREAMP_OFF_PATHGAIN_LOW	4	增益：前置放大关，通路增益低
AGC_MODE_AUTO	0	AGC 模式：自动

13.10 RX 分析器（检波方式）

DETECTOR_MAX	0	检波：最大值
DETECTOR_MIN	1	检波：最小值
DETECTOR_AVG	2	检波：平均值
DETECTOR_RMS	3	检波：均方根值
DETECTOR_Samp_oLE	4	检波：采样保持
DETECTOR_QP	5	检波：准峰值
DETECTOR_MAXPOWER	6	检波：最大功率
DETECTOR_AUTOPEAK	7	检波：自动峰值
DETECTOR_BYPASS	8	检波：旁路（不检波）

13.11 FFT 窗型

FFTWINDOW_NUTTALL	0	FFT 窗：Nuttall
FFTWINDOW_FLATTOP	1	FFT 窗：FlatTop
FFTWINDOW_LOWSIDELOBE	2	FFT 窗：低旁瓣
FFTWINDOW_GUASSIAN	3	FFT 窗：高斯

13.12 TX 模式

TX_CW	0	TX 模式：连续波
TX_REALTIME	1	TX 模式：实时播放
TX_PLAYBACK	2	TX 模式：波形回放

13.13 触发输入源

TRIGGER_SOURCE_BUS	0	总线触发
TRIGGER_SOURCE_EXTERNAL_RISING_EDGE	1	外部上升沿触发
TRIGGER_SOURCE_EXTERNAL_FALLING_EDGE	2	外部下降沿触发
TRIGGER_SOURCE_XPPS	3	GNSS 秒脉冲触发
TRIGGER_SOURCE_LEVEL	4	电平触发，暂不支持
TRIGGER_SOURCE_FMT	5	频谱模板触发，暂不支持
TRIGGER_SOURCE_TIMEDOMAIN_GATE	6	时域门触发，暂不支持

13.14 触发输出源

TRIGGER_IN_ROUTE	2	原始触发事件输出

13.15 触发输出脉冲极性

TRIGGER_PULSE_POSITIVE	0	正脉冲输出
TRIGGER_PULSE_NEGATIVE	1	负脉冲输出

13.16 触发通道行为

TRIGGER_ACTION_HOP	0	执行一次跳频
TRIGGER_ACTION_SWEEP	1	执行一次完整扫描

13.17 数字调制类型

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 滤波器类型

MOD_FILTER_RECTANGULAR	1	矩形滤波器
MOD_FILTER_RAISED_COSINE	2	升余弦滤波器
MOD_FILTER_ROOT_RAISED_COSINE	3	根升余弦滤波器
MOD_FILTER_GAUSSIAN	4	高斯滤波器
MOD_FILTER_HALF_SINE	5	半正弦滤波器

13.19 调制波形

MOD_SHAPE_SINE	1	正弦波
MOD_SHAPE_SQUARE	2	方波
MOD_SHAPE_TRIANGLE	3	三角波
MOD_SHAPE_RAMP	4	锯齿波

13.20 多音相位模式

MOD_MULTITONE_PHASE_FIXED	0	所有音使用固定初相 0 rad
MOD_MULTITONE_PHASE_RANDOM	1	使用 seed 生成可复现随机相位
MOD_MULTITONE_PHASE_PARABOLIC	2	使用二次相位律降低峰均比

附录 Appendix 1： API 返回值索引

错误代码	错误/警告原因	类型[1]	处理
0	无错误	-	无需处理，可正常执行后续过程。
-1	总线打开错误	错误	检查设备供电、数据线连接，检查驱动程序是否正确安装。排除错误后需要重新调用 device_open_usb 打开设备。
-8	总线传输错误	错误	重新插拔设备，若仍无法解决，请联系官方技术支持。
-29	发射功率校准文件丢失[2]	错误	检查发射功率校准文件是否放置在规定的目录下。排除错误后需要重新调用 device_open_usb 打开设备。
-43	IQ 校准文件丢失[2]	错误	检查 IQ 校准文件是否放置在规定的目录下。排除错误后需要重新调用 device_open_usb 打开设备。
10	数据传输超时	警告	等待 10ms 后，重新尝试下发。
42	使用默认发射功率校准文件	警告	不影响设备继续运行，但射频输出功率存在偏差。
44	使用默认 IQ 校准文件	警告	不影响设备继续运行，但精度会存在偏差。
98	参数超界发生钳位	警告	将越界参数强制修改回设备参数范围，可通过 query 查询当时的真实值，不影响设备继续运行。
99	发射功率过低，但未越界	警告	功率可能不准，不影响设备继续运行，请联系官方技术支持。
100	发射功率过高，但未越界	警告	功率可能不准，不影响设备继续运行，请联系官方技术支持。
15	系统时钟失锁	警告	测试结果可能不准，不影响设备继续运行，请联系官方技术支持。
16	ADC 时钟失锁	警告	测试结果可能不准，不影响设备继续运行，请联系官方技术支持。
18	射频本振失锁	警告	测试结果可能不准，不影响设备继续运行，请联系官方技术支持。
-601	调制许可证文件缺失	错误	请将 xx._mod.lic 文件放到/CalFile/文件夹中。
-602	调制许可文件验证失败	错误	通常为许可证内容错误，请联系官方技术支持。
-603	设备校验失败	错误	创建设备对象时，设备校验失败，请检查设备是否正常打开，并联系官方技术支持。
-604	对象创建失败	错误	创建设备对象时，对象创建失败，请检查设备是否正常打开，并联系官方技术支持。
-605	对象释放失败	错误	释放设备对象失败，请联系官方技术支持。
-606	调制类型错误	错误	生成信号时，输入调制类型错误，请修改调制类型并下发，重新生成信号。

-607	调制参数错误	错误	生成信号时，输入调制参数错误，请修改调制参数并下发，重新生成信号。
-608	缺少调制库	错误	h2_api 库的同路径下没有调制库，请将 genSignalWave.dll(Windwos)或 libgenSignalWave.so(Linux)放到 h2_api 库的同路径下。
-609	参数无法配置	错误	当前参数无法配置，已配置上一次可用参数。

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

[2] 对于返回值为-29、-43 的情况，还需要确认文件存放路径是否为全英文路径。若路径中包含中文字符，调用 API 时也会提示文件加载失败。

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