WT API 使用手册

Revision History

序号	版本	发布日期	修改描述
1	1.0	2014-08-02	WT 无线网络测试仪 API 初版
2	1.1	2014-10-16	删除旧的枚举定义
3	1.2	2014-11-04	添加使用指引
4	1.3	2014-12-03	兼容所有 200 系列的 WT 无线网络测试仪
5	1.4	2015-04-08	更新 API 版本说明
6	1.5	2015-06-08	添加 VSG 波形文件下载
7	1.6	2015-07-30	添加 ZigBee 分析选项
8	1.7	2015-10-20	添加重采样接口
9	1.8	2016-8-18	API GetResult 结果选项新增“PSDU LENGTH”
10	1.9	2016-12-28	添加多个 API 接口

<table>
<tr><td rowspan="3">WT API 使用手册</td><td>文档编号</td><td>版本</td></tr>
<tr><td>RD-0000-0003-0001</td><td>1.7</td></tr>
<tr><td colspan="2">Shenzhen iTest Technology Co., Ltd.</td></tr>
</table>

目录

WT API 使用手册

1 概述

该文档描述WLAN.Tester.API的编程使用注意事项，WLAN.Tester.API为WT-160无线网络测试仪、WT-200无线网络测试仪、WT-208无线网络测试仪 (以下简称WT无线网络测试仪)的操作接口，该接口以Dll的形式提供，可以包含于其他应用程序中使用。

1.1 引用

该接口包含以下文件资料:

1. 接口头文件, tester.h [1].
2. 枚举定义头文件, AnalyseDef.h， testerCommon.h [2].
3. WLAN.Tester.API.dll 及其附属的程序 [3].
4. API 使用 Demo 程序 [4].

2 系统应用

WLAN.Tester.API为使用者提供一套完整对WT无线网络测试仪进行控制、对WLAN信号分析的函数接口，以及函数的使用说明。

这些函数接口，从功能类型上分为以下几类：

(1) 设备连接

(2) 设备控制：获取或设置设备参数

(3) WLAN信号发送

(4) WLAN信号采集、分析、结果获取

注意:

该手册不包含对DUT的各种配置及其处理逻辑。

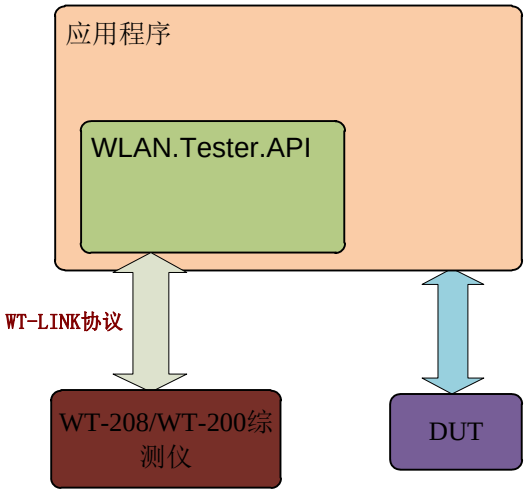


图 1 WLAN.Tester.API 应用场景

用户通过加载并调用WLAN.Tester.API.dll ,可以完成对WT无线网络测试仪的一系列控制，抓取数据之后，可以通过指定函数获取多种结果。

2.1 WLAN.Tester.API.dll

对于C/C++程序，需要使用以下文件：

WLAN.Tester.API.dll(包含其附属dll): WT无线网络测试仪程序接口dll

WLAN.Tester.API.lib: VC6等工具编译时需提供该引用库

tester.h: 描述WT无线网络测试仪程序接口的头文件

AnalyseDef.h，testerCommon.h: VSA,VSG参数中的枚举定义

注意：tester.h已经包含AnalyseDef.h,testerCommon.h

3 接口说明

3.1 Dll 初始化以及释放

在使用WLAN.Tester.API之前，需初始化该DLL的各种处理机制，并且在停止使用该DLL时，需释放该DLL所占的资源。

- WT_DLLInitialize
- WT_DLLTerminate

3.1.1 WT_DLLInitialize

原形	void WT_ DLLInitialize (void);
用途	初始化 WLAN.Tester.API 处理
备注	在调用 WLAN.Tester.API 其他 API 之前需调用该接口。该函数只能初始化一次
输入	void
输出	void

3.1.2 WT_DLLTerminate

原形	void WT_DLLTerminate(void);
用途	释放 WLAN.Tester.API 处理
备注	强烈建议在分析结束之后才调用该接口，否则可能造成其他内存处理出错，和初始化函数 WT_DLLInitilaze 配对使用
输入	void

输出	void

3.2 连接与断开

在通过WLAN.Tester.API控制WT无线网络测试仪之前，需连接到指定的WT无线网络测试仪，并且在完成测试项目之后，需断开对WT无线网络测试仪的连接。

- WT_Connect
- WT_DisConnect
- WT_ForceConnect

3.2.1 WT_Connect

原形	int WT_Connect(char *ipAddr, int *connID);
用途	连接指定 WT 无线网络测试仪
备注	在对 WT 无线网络测试仪进行各种操作之前，需先与其建立连接 ■ 该接口不会强制抢断指定 WT 无线网络测试仪与其他 EXE 之间的连接 ■ 多连接是 WT-208 无线网络测试仪新增的特征，在连接 WT-208 无线网络测试仪时，可以同时响应 1 ~ 4 个用户的连接以及后续操作请求(注：最多支持 4 个用户连接)，目前其他 WT 无线网络测试仪(WT160,WT200)还不支持该特性，只能建立一个连接；但当指定的 WT 无线网络测试仪已经被 WLAN Meter 连接上之后，即使是连接 WT-208 无线网络测试仪，也不会与该接口的调用程序创建新的连接。 ■ 连接失败时，可以通过 **WT_GetTesterConnStataus** 查询失败原因。
输入	**ipAddr**:指定 WT 无线网络测试仪的 IP 地址，如"192.168.10.254"

	connID:连接 ID 标识，通过该 ID 完成对此连接操作的识别
输出	■ 如果连接成功，则返回 WT_ERR_CODE_OK ■ 如果连接失败，则返回 WT_ERR_CODE_CONNECT_FAIL

3.2.2 WT_DisConnect

原形	void WT_DisConnect(int connID);
用途	断开连接，释放该连接所占用的所有资源
备注	在结束对指定 WT 无线网络测试仪的相应操作之后，通过该接口释放所占用的资源
输入	**connID**: 连接时获取的 ID 标识
输出	void

3.2.3 WT_ForceConnect

原形	int WT_ForceConnect(char *ipAddr, int *connID);
用途	强制连接仪器
备注	强制连接仪器会导致断开其他 IP 的连接 例如： A 机器 IP：192.168.10.100 B 机器 IP：192.168.10.200 WT 无线网络测试仪 IP：192.168.10.254 B 机器已经连接到 WT 无线网络测试仪，

	此时 A 机器调用 WT_ForceConnect 连接 WT 无线网络测试仪， 则 WT 无线网络测试仪会断开 B 的连接，A 强制占用 WT 无线网络测试仪 注意：此函数不支持多用户连接
输入	**ipAddr**:指定 WT 无线网络测试仪的 IP 地址，如"192.168.10.254" **connID**:连接 ID 标识地址，如果连接成功会保存连接 ID 标识符
输出	■ 如果连接成功，则返回 WT_ERR_CODE_OK ■ 如果连接失败，则返回 WT_ERR_CODE_CONNECT_FAIL

3.3 参数设置

在通过WLAN.Tester.API使用WT无线网络测试仪的过程中，需对各种参数进行设置以达到预期的效果。

- WT_SetExternalGain
- WT_SetVSA
- WT_SetVSG
- WT_GetDefaultParameter
- WT_SetVSA_AutoRange
- WT_SetWT208NetInfo
- WT_SetVSASampleRateMode
- WT_SetBandwidthMode

3.3.1 WT_SetExternalGain

原形	int WT_SetExternalGain(int connID, double extGain);

用途	设置 DUT 与指定 WT 无线网络测试仪之间的外部增益值
备注	需在连接之后才能设置外部增益值 该值对 WLAN.Tester.API 的 VSA、VSG 同时生效
输入	**connID**: 连接时获取的 ID 标识 **extGain**: 外部增益值，单位 dB， 注意:此处为外部增益，并非外部衰减。 ● 如果外部线衰减值是 10dB，则设定 extGain 的值为-10； ● 如果需要放大功率就需要设定 extGain 为正值。 强烈建议使用实际增益值，该值设置过大可能会导致 VSA、VSG 结果异常
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL

3.3.2 WT_SetVSA

原形	int WT_SetVSA(int connID, VsaParameter *vsaParam);
用途	设置 VSA 参数
备注	■ 需在连接之后才能设置 ■ vsaParam 中的 max_power 最好在进入仪器的实际功率基础上加 12dB ■ 如果接收功率范围未知，建议采用 WT_SetVSA_AutoRange 进行设置
输入	**connID**: 连接时获取的 ID 标识 **vsaParam**: VSA 详细参数 ***4.1.1***
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK

	■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果输入参数为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.3.3 WT_SetVSG

原形	int WT_SetVSG(int connID, VsgParameter *vsgParam);
用途	设置 VSG 参数
备注	需在连接之后才能设置 ■ 调用该参数之后如果指定的波形文件有效，会停止 VSG，需重新开启 VSG； ■ 如果指定的波形文件无效或者参数 vsgParam 为空，会返回 WT_ERR_CODE_UNKNOW_PARAMETER，且不影响之前的 VSG 状态
输入	**connID**: 连接时获取的 ID 标识 **vsgParam**: VSG 详细参数 ***4.1.2***
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果输入参数为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 如果指定的 tx 波形文件为空或者读取失败，则返回 WT_ERR_CODE_UNKNOW_PARAMETER，在该情况下除 VSG 的信号处理不更改外，其余参数均使用 VSG 参数中的最新值(如果不需要更改发送的信号时可以使用，并忽略该接口的返回值)

	■ 如果指定的波形文件过大，超出本机所剩内存，则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.3.4 WT_GetDefaultParameter

原形	int WT_GetDefaultParameter(VsaParameter *vsaParam, VsgParameter *vsgParam);
用途	填充默认的 VSA 参数到 vsaParam，VSG 参数到 vsgParam
备注	强烈建议在设置VSA，VSG之前先执行该操作，之后再在此基础上进行修改
输入	**vsaParam**: VSA 详细参数地址 ***4.1.1*** **vsgParam**: VSG 详细参数地址 ***4.1.2***
输出	■ 如果获取成功，则返回 WT_ERR_CODE_OK，同时将获取到的 VSA,VSG 参数分别更新到 vsaParam 以及 vsgParam(wave,repeat 以及 wave_gap 部分不会更新)中 ■ 如果指定的参数为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER

3.3.5 WT_SetVSA_AutoRange

原形	int WT_SetVSA_AutoRange(int connID, VsaParameter *vsaParam);
用途	根据 VSA 参数 ,自动匹配参考电平值(Max Power Level),并在匹配成功之后配置到 WT 无线网络测试仪中去
备注	强烈建议在输入信号未知的前提下，优先使用该接口 需在连接之后才能设置

	■ 如果输入到 WT 无线网络测试仪中的信号间隔(IFG)大于 200ms ,或者信号功率跳变非常大时，可能会导致 Auto Range 返回的参考电平不准确
输入	**connID**: 连接时获取的 ID 标识 **vsaParam**:vsa 详细参数 ***4.1.1***，除参考电平值(Max Power Level)、触发模式(trig_type)、采样时间(smp_time)外，其余设置均采用指定的值
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK，同时将新的 VSA 参数设置到指定的 WT 无线网络测试仪中去，并将新的参考电平值填充到指定的 VSA 参数中 ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果指定的参数为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 如果无足够内存处理数据分析 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.3.6 WT_SetWT208NetInfo

原形	int WT_SetWT208NetInfo(int connID, WT_208_NET_TYPE *ipAddrs);
用途	设置 WT-208 的几个子网口使用信息。具体用例参考 ***5.3.4***
备注	需在连接之后才能设置 子网口配置是 WT-208 新增的特征，其主要作用是解决在同一子网内多个 DUT 的 IP 冲突问题，可以使用同一 PC 控制相同 IP 的多个 DUT 进行测试。DUT IP, 必须与仪器 IP 不在同一网段，与虚拟 IP 也不能同一个网段 子网口配置共分三种情况如下（2 和 3 不会同时存在）:

	■ 场景 1，无需 TFTP 时的配置： **DUT 实际 IP 为**：192.168.1.1 **虚拟 IP 配置为**：192.168.100.1～192.168.100.4 DUT_1~DUT_4 分别映射到虚拟 IP_1~IP_4；PC 连接 DUT 时，使用 DUT 所映射的虚拟 IP 地址。注意虚拟 IP 不能同 DUT 实际 IP 在同一个网段 ■ 场景 2，DUT 作为 TFTP 服务器的配置： 在**“场景 1”**的基础上，添加下面的配置项： **DUT 指定的 Server IP**：192.168.1.100 PC 可通过虚拟 IP_1~IP_4 分别访问搭建在 DUT 上的 TFTP 服务器 ■ 场景 3，PC 作为 TFTP 服务器的配置： 在**“场景 1”**的基础上，添加下面的配置项： **DUT 指定的 Server IP**：192.168.2.100 **DUT 在 TFTP 时指定的 Client IP**：192.168.2.1(必须与 Server IP 不一致) **PC 作为 Server 的实际 IP**：192.168.100.100 (必须与虚拟 IP1~4 在同一网络中) 通过这种方式，DUT 便可从 PC 端下载文件数据
输入	**connID**: 连接时获取的 ID 标识 **ipAddrs**:各网口的配置信息
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果指定的参数为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER

3.3.7 WT_SetVSASampleRateMode

原形	int WT_SetVSASampleRateMode(int connID, int sampleRateMode)
用途	用于设置 VSA 采样速率。API 2.7.4.32 支持 WIFI,BT 可以 120M/60M 重采样，ZigBee 只支持 120M 采样率
备注	API 版本小于 2.7.4.32 不支持此接口 需在连接仪器之后才能操作。支持该功能的固件版本必须大于： WT-200B: 2.2.1.30 WT-200: 2.2.0.29 WT-208: 3.0.1.13 WT-160:不支持
输入	**connID**: 连接时获取的 ID 标识 **sampleRateMode**: 采样模式 enum TESTER_SAMPLERATE_MODE(*4.3.24*)。分别为默认和自动两种模式，对应的值为 **RATE_DEFAULT，RATE_AUTO。**其中 RATE_AUTO 表示 WIFI,BT 根据 VSA 配置自动调整采样速率 ,RATE_DEFAULT 表示使用默认采样速率(WT-20X 120M,WT-160 80M)
输出	■ 如果成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ WT_ERR_CODE_UNKNOW_PARAMETER: 参数配置有误 ■ WT_ERR_CODE_GENERAL_ERROR: 配置失败，仪器不支持该功能

3.3.8 WT_SetBandwidthMode

原形	int WT_SetBandwidthMode (int connID, int bandwidthDetect)
用途	设置带宽识别模式
备注	
输入	**connID**: 连接时获取的 ID 标识 **bandwidthDetect**: 带宽模式，参考 WT_BANDWIDTH_DETECT_ENUM（*4.3.18*）枚举
输出	■ 如果成功，则返回 WT_ERR_CODE_OK

3.4 信号抓取以及分析

在通过WLAN.Tester.API使用WT无线网络测试仪的过程中，需对接入到WT无线网络测试仪中的WLAN信号进行分析以判断DUT的信号发送质量。

- WT_DataCapture
- WT_StopDataCapture
- WT_GetResult
- WT_GetVectorResult
- WT_GetVectorResultElementSize
- WT_GetVectorResultElementCount

3.4.1 WT_DataCapture

原形	int WT_DataCapture(int connID);

用途	按照设定的 VSA 参数进行数据抓取
备注	在对输入信号进行分析之前，需调用该接口抓取数据 需在连接之后才能调用
输入	**connID**: 连接时获取的 ID 标识
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果无足够内存装载指定数据 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果操作超时，则返回 WT_ERR_CODE_TIMEOUT -如果处理中出现其他错误，则返回WT_ERR_CODE_GENERAL_ERROR

3.4.2 WT_StopDataCapture

原形	int WT_StopDataCapture(int connID);
用途	停止 VSA 数据抓取
备注	
输入	**connID**: 连接时获取的 ID 标识
输出	■ 如果设置成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果无足够内存装载指定数据 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果操作超时，则返回 WT_ERR_CODE_TIMEOUT ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.4.3 WT_GetResult

原形	int WT_GetResult(int connID, char *anaParamString , double *result, char *description, char *unit);
用途	获取指定 anaParamString 的分析结果
备注	■ 获取成功后，各输出参数值才有效 ■ 需在 WT_DataCapture 抓取到相应信号数据之后才能使用 ■ 可以对(通过 WT_DataCapture 获取的)同一数据进行多次分析
输入	**connID**: 连接时获取的 ID 标识 **anaParamString**: 分析项，例如 WT_RES_EVM_ALL，详见分析参数定义 **result**:装载分析结果的地址 **description**：装载结果描述的容器地址 **unit**:装载单位描述的容器地址
输出	■ 如果分析成功 ,则返回 WT_ERR_CODE_OK ,同时(在结果容器非空的前提下)将获取到的结果、结果描述、单位分别装载到 result、description、unit 中 ■ 如果在此之前未抓取信号数据，则返回 WT_ERR_CODE_NO_DATA_CAPTURED ■ 如果指定的分析类别参数非法，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 如果无足够内存处理数据分析 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.4.4 WT_GetVectorResult

原形	int WT_GetVectorResult(int connID, char *anaParamString , void *result, int resultType, int resultSize);
用途	获取指定 anaParamString 的分析结果
备注	■ 获取成功后，各输出参数值才有效 ■ 需在 WT_DataCapture 抓取到相应信号数据之后才能使用 ■ 可以对(通过 WT_DataCapture 获取的)同一数据进行多次分析
输入	**connID**: 连接时获取的 ID 标识 **anaParamString**: 分析项，例如 WT_RES_EVM_ALL，详见分析参数定义 **result**: 对应结果存储区域 **resultType**：结果的数据类型，参考 WT_RESULT_TYPE **resultSize**: 结果存储区域大小，单位：字节
输出	■ 如果分析成功，返回实际复制到结果的数目，单元大小以输入的 resultType 为准 ■ 返回 0 代表参数错误或者分析失败

3.4.5 WT_GetVectorResultElementSize

原形	int WT_GetVectorResultElementSize (int connID, char *anaParamString , int *elementSize);
用途	获取结果中每个单元数据的大小，单位：字节
备注	■ 获取成功后，各输出参数值才有效

	■ 需在 WT_DataCapture 抓取到相应信号数据之后才能使用 ■ 可以对(通过 WT_DataCapture 获取的)同一数据进行多次分析
输入	**connID**: 连接时获取的 ID 标识 **anaParamString**: 分析项，例如 WT_RES_EVM_ALL，详见分析参数定义 **elementSize**:装载“输出单元数据的大小”的地址
输出	■ 如果分析成功 ,则返回 WT_ERR_CODE_OK ,同时(在结果容器非空的前提下)将获取到的结果、结果描述、单位分别装载到 result、description、unit 中 ■ 如果在此之前未抓取信号数据，则返回 WT_ERR_CODE_NO_DATA_CAPTURED ■ 如果指定的分析类别参数非法，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 如果无足够内存处理数据分析 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.4.6 WT_GetVectorResultElementCount

原形	int WT_GetVectorResultElementCount (int connID, char *anaParamString , int *elementCount);
用途	获取整个结果的长度
备注	■ 获取成功后，各输出参数值才有效 ■ 需在 WT_DataCapture 抓取到相应信号数据之后才能使用 ■ 可以对(通过 WT_DataCapture 获取的)同一数据进行多次分析
输入	**connID**: 连接时获取的 ID 标识

	anaParamString: 分析项，例如 WT_RES_EVM_ALL，详见分析参数定义 **elementCount**:输出结果的长度存放地址
输出	■ 如果分析成功 ,则返回 WT_ERR_CODE_OK ,同时(在结果容器非空的前提下)将获取到的结果、结果描述、单位分别装载到 result、description、unit 中 ■ 如果在此之前未抓取信号数据，则返回 WT_ERR_CODE_NO_DATA_CAPTURED ■ 如果指定的分析类别参数非法，则返回 WT_ERR_CODE_UNKNOW_PARAMETER ■ 如果无足够内存处理数据分析 则返回 WT_ERR_CODE_OUT_OF_MEMORY ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.5 信号发送以及停止

在通过WLAN.Tester.API使用WT无线网络测试仪的过程中，需通过WT无线网络测试仪发送指定WLAN信号以测试DUT的信号接收能力。

- WT_StartVSG
- WT_AsynStartVSG
- WT_StopVSG
- WT_GetVSGCurrentState

3.5.1 WT_StartVSG

原形	int WT_StartVSG(int connID);
用途	按照指定的 VSG 参数进行信号(wave)发送，并在发送完成后才返回(同步进行)

备注	■ 需在连接之后才能设置 ■ 一般测试 DUT 的 PER 时，都采用该方式进行 ■ 如果无其他线程控制 VSG 结束，指定的发送次数为无限次(0)，则该函数在网络断开之前不会返回
输入	**connID**: 连接时获取的 ID 标识
输出	■ 如果开启成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果操作超时，则返回 WT_ERR_CODE_TIMEOUT ■ 如果 VSG 功率不准确，则返回 WT_ERR_CODE_VsgInaccuracy ■ 如果 VSG 处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.5.2 WT_AsynStartVSG

原形	int WT_AsynStartVSG(int connID);
用途	按照指定的 VSG 参数进行信号(wave)发送，并在命令下发后直接返回(异步进行)
备注	需在连接之后才能调用
输入	**connID**: 连接时获取的ID标识
输出	■ 如果开启成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果操作超时，则返回 WT_ERR_CODE_TIMEOUT ■ 如果 VSG 功率不准确，则返回 WT_ERR_CODE_VsgInaccuracy

	■ 如果 VSG 处理中出现其他错误,则返回 WT_ERR_CODE_GENERAL_ERROR

3.5.3 WT_StopVSG

原形	int WT_StopVSG(int connID);
用途	停止 VSG 发送
备注	■ 无论何时调用此接口，或是指定 WT 无线网络测试仪是否正在发送，执行步骤成功之后，都会停止发送 ■ 需在连接之后才能调用
输入	**connID**: 连接时获取的 ID 标识
输出	■ 如果停止 VSG 成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.5.4 WT_GetVSGCurrentState

原形	int WT_GetVSGCurrentState (int connID, int *state);
用途	获取 VSG 的当前状态，当开启异步 VSG，且需要了解当前 VSG 状态时可用。
备注	需在连接之后才能调用
输入	**connID**: 连接时获取的 ID 标识 **state:** 获取到的VSG状态，详见WT_VSG_STATE
输出	■ 如果获取成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回

	WT_ERR_CODE_CONNECT_FAIL ■ 如果操作超时，则返回 WT_ERR_CODE_TIMEOUT ■ 如果处理中出现其他错误，则返回 WT_ERR_CODE_GENERAL_ERROR

3.6 信息查询

在通过WLAN.Tester.API使用WT无线网络测试仪的过程中，可以通过指定接口访问对应的WT无线网络测试仪版本信息以及连接状态。

- WT_GetTesterVersion
- WT_GetTesterConnStataus
- WT_GetTesterSpecification
- WT_GetSymbolsEvm
- WT_DataStore

3.6.1 WT_GetTesterVersion

原形	int WT_GetTesterVersion(int connID, TesterInfo *buffer, int bufferSize, int *testerType);
用途	获取当前 WT 无线网络测试仪版本信息以及 WT 无线网络测试仪类型
备注	需在连接之后才能获取
输入	**connID**: 连接时获取的 ID 标识 **buffer**: 装载 WT 无线网络测试仪版本信息的容器，查询成功后，装载结果详见 TesterInfo **bufferSize**: 指定容器的大小

	testerType:仪器类型
输出	■ 如果查询成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果指定的容器为空或者容器大小不够，或者指定 WT 无线网络测试仪未连接，则返回 WT_ERR_CODE_UNKNOW_PARAMETER

3.6.2 WT_GetTesterConnStataus

原形	int WT_GetTesterConnStataus(char *ipAddr, char *buffer, int bufferSize);
用途	获取 WT 无线网络测试仪当前的连接状态
备注	■ 可以在没有连接 WT 无线网络测试仪的情况下获取指定 WT 无线网络测试仪的连接状态 ■ 在连接仪器 WT 无线网络测试仪失败时，可用通过该接口查看 WT 无线网络测试仪是否能连接，或者是否正被其他 EXE 所占用。 针对以太网的使用，设备存在以下 3 种状态： ■ 忙状态 **BUSY:IP:192.168.10.2;PORT:28691** 表示设备正被 192.168.10.2 的 28691 端口使用 ■ 空闲状态 **IDLE:** 表示设备处于空闲状态，可用连接 ■ 设备不存在

	ERR:DEVICE DOES NOT EXIST 无法连接 ipAddr 指定的设备
输入	**ipAddr:** 装载目标 WT 无线网络测试仪 IP，如"192.168.10.254" **buffer:** 返回操作结果，字符串，由 WLAN.Tester.API 生成并返回 **bufferSize:**buffer 的大小
输出	■ 如果网络无法连接，则返回 0 ■ 如果接口成功执行，则返回 buffer 数据的有效长度

3.6.3 WT_GetTesterSpecification

原形	int WT_GetTesterSpecification (int connID, DeviceSpecification *testerSpec);
用途	获取当前 WT 无线网络测试仪版本信息以及 WT 无线网络测试仪类型
备注	需在连接之后才能获取
输入	**connID**: 连接时获取的 ID 标识 **testerSpec**:Tester 规格存放地址
输出	■ 如果查询成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 如果 testerSpec 为空，则返回 WT_ERR_CODE_UNKNOW_PARAMETER

3.6.4 WT_GetSymbolsEvm

原形	int WT_GetSymbolsEvm(int connID, int startSymbol, int endSymbol, int evmResultType, double *evmResult);
用途	获取某段 Symbol 范围内的 Evm 均值
备注	API 版本低于 2.7.5.42 不支持此接口
输入	**connID**: 连接时获取的 ID 标识 **startSymbol:** 起始 Symbol 序号，从 0 开始 **endSymbol:** 结束 Symbol 序号，计算时不包含该 Symbol. **evmResultType**: 设定取哪种类型的 evm 结果，参照 enum SYMBOLS_EVM_RESULT_TYPE（***4.3.25***）. **evmResult(Out):** 存储规定 Symbol 范围内的 Evm 均值的缓存，单位 dB.
输出	■ 如果获取成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL

3.6.5 WT_DataStore

原形	int WT_DataStore(int connID, char *fileName);
用途	保存 Vsa 信号数据
备注	API 版本低于 2.7.5.42 不支持此接口
输入	**connID**: 连接时获取的 ID 标识 **fileName:** 保存文件名（包含路径）

输出	■ 如果查询成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL

3.7 VSG 波形文件配置

在通过 WLAN.Tester.API 使用 WT 无线网络测试仪的过程中，可以通过指定 VSG 使用的波形文件。目前支持三种模式：PC 本地波形文件，仪器默认波形文件，仪器存储的用户自定义波形文件。对波形文件提供下面几种操作：添加，删除，查询。具体使用 demo 请查看 *5.6*

注意：

- WT-208 固件版本低于 3.0.1.13 不支持该项
- WT-200 固件版本低于 2.2.1.27 不支持该项
- WT-160 不支持该项
- 提供了 1GB 的存储空间给自定义信号文件

3.7.1 WT_OperateTesterWave

原形	int WT_OperateTesterWave (int connID, char *waveName, int operateOption);
用途	用于添加信号到仪器或者删除仪器的信号
备注	需在连接仪器之后才能操作
输入	**connID**: 连接时获取的 ID 标识 **waveName**: 信号文件名或者波形文件路径

	operateOption：具体操作选项。有“ADD_WAVE”，“REMOVE_WAVE”两种操作。
输出	■ 如果成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 返回 WT_ERR_CODE_GENERAL_ERROR: waveName 为空或者其他错误 ■ WT_ERR_CODE_UNKNOW_PARAMETER: 参数配置有误

3.7.2 WT_QueryTesterWave

原形	int WT_ QueryTesterWave (int connID, char *waveName, int *testerExistWave);
用途	用于查询仪器是否存在指定的波形文件
备注	需在连接仪器之后才能操作
输入	**connID**: 连接时获取的 ID 标识 **waveName**: 信号文件名或者波形文件路径 **testerExistWave：**存储查询结果的容器地址。返回值 1：仪器存在指定的波形文件；返回值 0：仪器不存在指定的波形文件。
输出	■ 如果成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 返回 WT_ERR_CODE_GENERAL_ERROR: waveName 为空或者其他错误 ■ WT_ERR_CODE_UNKNOW_PARAMETER: 参数配置有误

3.7.3 WT_GetTesterAllWaveNames

原形	int WT_GetTesterAllWaveNames (int connID, char *fileNameBuffer, int fileNameBufferSize, int *fileCount);
用途	获取仪器中的信号文件列表
备注	需在连接仪器之后才能操作
输入	**connID**: 连接时获取的 ID 标识 **fileNameBuffer**: 用于存储仪器信号文件名列表的容器地址 ,相邻两个信号文件名以**"\r\n"**隔开 **fileNameBufferSize：**fileNameBuffer 的大小，单位：字节。 **fileCount**: 用于存储仪器信号文件数量的容器地址 (注：如果 fileNameBuffer 太小则不能完整存储仪器中所有的信号文件名)
输出	■ 如果成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ 返回 WT_ERR_CODE_GENERAL_ERROR: fileNameBuffer 为空或者其他错误 ■ WT_ERR_CODE_UNKNOW_PARAMETER: 参数配置有误

3.8 Beamforming 测试接口

目前 API 支持的 Bcm4360 的 Beamforming 测试。共用校准和验证两个部分。

校准相关 API:

WT_BeamformingCalibrationChannelEstDutTX

WT_BeamformingCalibrationChannelEstDutRX

WT_BeamformingCalibrationResult

WT_BeamformingCalculateChannelProfile

验证相关 API:

WT_BeamformingVerification

3.8.1 WT_BeamformingCalibrationChannelEstDutTX

原形	int WT_BeamformingCalibrationChannelEstDutTX(int connID, int demod)
用途	在Calibration时，控制DUT发送，WT-200接收，调用此函数估算出Hab 1x3
备注	
输入	**connID:** 连接时获取的 ID 标识. **demod:** 枚举量 802.11 分析模式.WT_DEMOD_ENUM
输出	WT_ERR_OK:操作成功 其它:分析失败

3.8.2 WT_BeamformingCalibrationChannelEstDutRX

原形	int WT_BeamformingCalibrationChannelEstDutRX(int connID, double *dutChannelEst, int dutChannelEstLength)
用途	在Calibration时，控制WT-200发送，DUT接收，调用此函数估算出Hba 3x1
备注	

输入	**connID:** 连接时获取的ID标识. **dutChannelEst:** DUT读取到的通道信息数组 **dutChannelEstLength:** DUT读取到的通道信息数组长度
输出	WT_ERR_OK:操作成功 其它:分析失败

3.8.3 WT_BeamformingCalibrationResult

原形	int WT_BeamformingCalibrationResult(int connID, double *resultAngle, int *resultAngleLength)
用途	在Calibration时，在获取Hab 1x3和Hba 3x1后，通过此函数获取相位
备注	
输入	**connID:** 连接时获取的ID标识. **resultAngle:** 结果相位值，返回为数组（返回最大长度8元素），由调用方提供数组内存 **resultAngleLength:** 结果相位值数组长度
输出	WT_ERR_OK:操作成功 其它:分析失败

3.8.4 WT_BeamformingVerification

原形	int WT_BeamformingVerification(int connID, double *diffPower)
用途	在Verification时，通过配置DUT进入相应状态并输出Beamforming信号，

	WT-200接收并解析DUT信号，计算Beamforming带来的增益。
备注	实现原理见下面说明
输入	**connID:** 连接时获取的ID标识 **diffPower:** 返回结果，Beamforming增益dB
输出	WT_ERR_OK:操作成功 其它:分析失败

3.8.5 WT_BeamformingCalculateChannelProfile

原形	int WT_BeamformingCalculateChannelProfile (int connID, int demod, double *resultBuf, int resultLength)
用途	在Calibration时，通过此函数获取DUT发送信号的幅度和相位，此函数适用于MTK方案
备注	API 版本小于 2.7.6.55 不支持此接口
输入	**connID:** 连接时获取的ID标识 **demod:** 运行场景枚举，详见： enum WT_DEMOD_ENUM (BW,HT/VHT) **resultBuf:** 结果相位值，返回为数组（幅度、相位），由调用方提供数组内存 **resultLength:** 结果值数组长度，必须是2的倍数
输出	WT_ERR_OK:操作成功 其它:分析失败

3.9 ZigBee 专用接口

3.9.1 WT_ SetZigBeeAnalysisOptimise

原形	int WT_SetZigBeeAnalysisOptimise(int connID, int AnalysisOptimise);
用途	开启/关闭 ZigBee分析优化选项
备注	API版本低于 2.7.6.49不支持此接口
输入	**connID:** 连接时获取的ID标识 **AnalysisOptimise:** 开关标识，关闭为：0；开启为：1；
输出	WT_ERR_OK:操作成功 其它:分析失败

3.10 VSA IFG 配置接口

API 2.7.3.28 之前的版本 AutoRange 只支持 waveform gap 小于 200ms 的信号。现在可以根据实际的 waveform gap 值来调整 VSA IFG 超时时间。

建议：一般情况下使用默认的 200ms 超时时间，当确认 waveform gap 值大于 200ms 且使用默认的 200ms Autorange 失败才建议修改。例如有些 ZigBee 的 waveform gap 值大于 200ms，可以设置 VSA waveform gap 超时时间为 500ms

3.10.1 WT_SetMaxIFG

原形	int WT_SetMaxIFG(int connID, int maxgap)
用途	用于设置 VSA AutoRange trigger 超时时间等于 VSA IFG
备注	需在连接仪器之后才能操作

输入	**connID**: 连接时获取的 ID 标识 **maxgap**: 超时时间，单位：ms 毫秒，范围：1-10000
输出	■ 如果成功，则返回 WT_ERR_CODE_OK ■ 如果未与指定的 WT 无线网络测试仪建立连接，则返回 WT_ERR_CODE_CONNECT_FAIL ■ WT_ERR_CODE_UNKNOW_PARAMETER: 参数配置有误

4 数据结构以及枚举定义

在WLAN.Tester.API中，提供简单的数据结构以及枚举定义，使得用户可以很方便地使用WLAN.Tester.API提供的各种功能。

4.1 数据结构定义

4.1.1 VSA 参数结构

```
//对应的一些IQxel-M参数，请看5.7

typedef struct

{

   //VSA载波频率,单位：Hz

   double  freq;

   //输入的最大功率,dBm; 最好在实际输出到仪器的功率基础上再加18dB
```

```
//如果接收功率范围未知，建议事先Auto Range
double  max_power;
//采样时间,us
double  smp_time;
//指定VSA使用的RF端口
int     rfPort;
//触发模式,WT_TRIG_TYPE_ENUM
int     trig_type;
//触发电平,与max_power的差距,dB
double  trig_level;
//触发等待超时,sec（该时间与TimeoutWaiting的时间相互独立）
double  trig_timeout;
//触发保留时间,保留触发前若干时间内的数据,单位us。
//wifi建议20us，BT建议150us
double  trig_pretime;
//802.11  分析模式.WT_DEMOD_ENUM
int     demod;
//IQ交换(频谱反转). WT_IQ_SWAP_ENUM
```

```
int         iq_swap;

//802.11 a/g/n/ac  相位跟踪. WT_PH_CORR_ENUM

int         ph_corr;

//802.11 a/g/n/ac  通道估计. WT_CH_EST_ENUM

int         ch_estimate;

//802.11 a/g/n/ac  时序跟踪. WT_SYM_TIM_ENUM

int         sym_tim_corr;

//802.11 a/g/n/ac  频率同步. WT_FREQ_SYNC_ENUM

int        freq_sync;

//802.11 a/g/n/ac  幅度跟踪. WT_AMPL_TRACK_ENUM

int        ampl_track;

//802.11 b EVM方式.    WT_11B_METHOD_ENUM

int         evm_method_11b;

//802.11 b  直流去除. WT_DC_REMOVAL_ENUM

int         dc_removal;

//802.11 b  均衡类型. WT_EQ_ENUM

int        eq_taps;

//802.11 b  相位跟踪. WT_PH_CORR_11b_ENUM
```

```
    int        cck_ph_corr;
    //Bluetooth速率, WT_BT_DATARATE
    int        bt_Rate;
    //默认设置为WT_BT_PACKETTYPE_NULL
    int        bt_packet_type;
    //在多连接情况下，等待的最大时间，默认值为8秒，单位sec
    int        TimeoutWaiting;
}VsaParameter;
```

注意：如果测试仪器室WT-208时，建议trig_timeout设置值要小于TimeoutWaiting，否则多连接测试情况下可能会导致WT_DataCapture返回WT_ERR_CODE_TIMEOUT

4.1.2 VSG 参数结构

```
typedef struct
{
    double  freq;                   //VSG载波频率,单位：Hz
    double  power;                  //VSG功率,dBm
    int        rfPort;              //指定VSG使用的RF端口
    int        waveType;            //Wave类型，详解请看下面注解
```

```
    char      *wave;                //VSG数据文件，详解请看下面注解
    int       repeat;               //循环发送次数
    double    wave_gap;             //两次发送间隔，us
    int       TimeoutWaiting;       //在多连接情况下，等待的最大时间，默认值为8秒
}VsgParameter;
```

VsgParameter结构waveType,wave详解：

- int waveType:

  Wave类型，枚举WT_SIGNAL_ENUM(***4.3.17***)。注意下面几点：

  - SIG_USERFILE: 读取本地wave文件。使用此参数，**VsgParameter**的wave参数必须不能为NULL，且wave路径必须是绝对路径;
  - SIG_TESTERFILE: 目前只有WT-200B支持 ,使用仪器存储的用户自定义wave文件(注意：***0***); 使用此参数，**VsgParameter**的wave参数不能为NULL，wave设置成对于的wave文件名而不是绝对路径
  - SIG_USERFILE和SIG_TESTERFILE以外的选项 :仅WT-208仪器支持。使用此参数，**VsgParameter**的wave参数设置为NULL，表示使用WT-208内部默认的信号文件。

- char *wave;

  - VSG数据文件,如果为NULL,只更新设备配置，不更新wave文件，不配置repeat和wave_gap

详细配置例子参考查看*5.6*

4.1.3 网口配置参数结构

```
typedef struct
{
  //*****************************************************************************
  //                                基础配置，该部分必须配置
  //*****************************************************************************

    char VirAddr1[16];      // 虚拟IP 1
    char VirAddr2[16];      // 虚拟IP 2
    char VirAddr3[16];      // 虚拟IP 3
    char VirAddr4[16];      // 虚拟IP 4
    // DUT IP,必须与仪器IP不在同一网段，与虚拟IP也不能同一个网段
    char DutAddr[16];

    //*****************************************************************************
    //   DUT作为TFTP Server部分
    // 无TFTP需求时，以下IP都设置为空
    // 需配置DUT在TFTP状态下指定的Server IP
    //*****************************************************************************
```

```
    char DutTftpServerAddr[16];                 // DUT在TFTP状态下指定的Server IP
    //*****************************************************************************
    //      FFTP部分
    //      无TFTP需求时，以下IP都设置为空
    //     当PC作为TFTP Server时，需配置实际作为Server的PC IP
    //     还需配置DUT在TFTP状态下所使用的Client IP以及DUT所指定的Server IP
    //*****************************************************************************
    // DUT在TFTP状态下所指定的Server IP
    char TftpServerAddr[16];
    // DUT在TFTP状态下所使用的Client IP,必须与TftpServerAddr不一样
    char TftpClientAddr[16];
    // PC作为TFTP Server时实际PC IP,需确保该IP与DUT的虚拟IP在同一网络下
    char TftpServerPCAddr[16];
}WT_208_NET_TYPE;
```

4.1.4 设备信息结构

```
typedef struct
{
    char ip[16];                            // IP地址
    char SubMask[16];                       // 子网掩码
```

```
    char GateAddr[16];                  // 网关
    char SN[80];                        // SN码
    char name[40];                      // 别名        "iTest Tester"
    char MacAddr[18];                   // MAC 地址  "DC-A4-B5-C7-E1-F8"
    char FW_Version[40];                    // 固件版本
    char HW_Version[40];                    // 硬件版本
    char RF_Version[40];                // 射频板版本
    char cal_date[20];                      // 校准日期  "2014-06-04 18:36:56"
} TesterInfo;
```

4.1.5 较线参数结构

```
typedef struct
{
    double  freq;               //中心频率,单位：Hz
    int     vsa_port;           //接收端口，WT_PORT_ENUM
    int     vsg_port;           //发送端口，WT_PORT_ENUM
    int     waveType;           //Wave类型，详细请看4.1.2中的waveType
    char    *wave;              //VSG数据文件，详细请看4.1.2中的wave
```

```
    int       TimeoutWaiting;       //在多连接情况下，等待的最大时间，单位sec
                                    //默认值为8秒，单位为：秒
} CableVerifyParameter;
```

4.1.6 设备规格结构

```
typedef struct
{
    char lic_5g;                // 是否支持5G：1支持；其他不支持
    char lic_ac;                // 是否支持ac测试
    char lic_bluetooth;         // 是否支持BT测试
    char lic_mimo;              // 是否支持MIMO测试
    char lic_zigbee;            // 是否支持Zigbee测试
    char temp[59];              // 保留位
}DeviceSpecification;
```

4.2 分析参数选项

```
//*****************************************************************************
//                                SPECTRUM
//*****************************************************************************
```

```
//Carrier leakage in dB.Value is returned
#define WT_RES_SPECTRUM_CARR_LEAKAGE                    "Spec_carrier_leakage"

//OBW(99%) in Hz.Value is returned
#define WT_RES_SPECTRUM_OBW_99                          "Spec_Obw"

//Spectrum mask error point in %.Value is returned
#define WT_RES_SPECTRUM_MASK_ERR_PERCENT                "Spec_mask_err"

//Frequency of the max power in Hz.Value is returned
#define WT_RES_SPECTRUM_PEAK_POW_FREQ                   "Spec_peak_freq"

//Spectrum data. Vector is returned.
#define WT_RES_SPECTRUM_DATA                            "Spec_data"

//Spectrum Mask Margin data. Complex Vector is returned.
#define WT_RES_SPECTRUM_MARGIN                          "Spec_margin"

//Frequency in Hz.Value is returned.
//VSA sampling time should be 10ms when analyze this. CW only.
#define WT_RES_CW_FREQ_OFFSET                           "CW_spec_freq_offset"
//*************************************************************************
```

```
//                                  IQ
//*****************************************************************************
//RAW data. Complex vector is returned.
#define WT_RES_RAW_DATA                                 "Raw_data"
//*****************************************************************************
//                                  POWER
//*****************************************************************************
// Frame Count by Power Detectoring. Value is returned.
#define WT_RES_POWER_FRAME_CNT                          "Pow_frame_count"

// Power frame (no gap) in dB. Value is returned.
#define WT_RES_POWER_FRAME_DB                           "Pow_frame"

// RMS Power in dB. Value is returned.
#define WT_RES_POWER_ALL_DB                             "Pow_all"

// Power Peak in dB. Value is returned.
#define WT_RES_POWER_PEAK_DB                            "Pow_peak"

//*****************************************************************************
//                                  WIFI Frame
//*****************************************************************************
```

```
// EVM for entire frame in dB. Value is returned.
#define WT_RES_FRAME_EVM_ALL                    "evm.all"

// EVM peak value in dB. Value is returned.
#define WT_RES_FRAME_EVM_PEAK                   "evm.pk"

// Frequency Error in Hz. Value is returned.
#define WT_RES_FRAME_FREQ_ERR                   "signal.freqerr"

// Symbol Clock Error in ppm. Value is returned.
#define WT_RES_FRAME_SYMBOL_CLOCK_ERR           "signal.symclockerr"

// IQ Match Amplitude Error in dB. Value is returned.
#define WT_RES_FRAME_IQ_MATCH_AMP               "iqmatch.amp"

// IQ Match Phase Error in Deg. Value is returned.
#define WT_RES_FRAME_IQ_MATCH_PHASE             "iqmatch.phase"

// IQ Phase Error in Deg. Value is returned.
#define WT_RES_FRAME_PHASE_ERROR                "phase.error"

// Data rate in Mbps. Value is returned.
```

```
#define WT_RES_FRAME_DATA_RATE_MBPS                    "Data_rate_Mbps"

// Ramp on time. Value is returned.
#define WT_RES_FRAME_RAMP_ON_TIME                      "ramp.on_time"

// Ramp off time. Value is returned.
#define WT_RES_FRAME_RAMP_OFF_TIME                     "ramp.off_time"

// Number of symbols. Value is returned. OFDM only
#define WT_RES_FRAME_OFDM_NUMBER_SYMBOLS
"ofdm.more_res.PLCP.Nspp"

// EVM for data part of frame.dB. Value is returned. OFDM only
#define WT_RES_FRAME_EVM_DATA_DB                       "evm.data"

// EVM for pilot part of frame.dB. Value is returned. OFDM only
#define WT_RES_FRAME_EVM_PILOT_DB                      "evm.pilot"

//Spectral flatness Passed Or Failed. 1(1.0) - Passed; 0(0.0) - Failed. OFDM only
#define WT_RES_SPECTRUM_FLATNESS_PASSED                "flatness.passed"

//Spectral flatness section value. Complex Vector is returned.   OFDM only
```

```
#define WT_RES_SPECTRUM_FLATNESS_SECTION_VALUE

"flatness.section.value"



//Spectral flatness section margin. Complex Vector is returned. OFDM only

#define WT_RES_SPECTRUM_FLATNESS_SECTION_MARGIN

"flatness.section.margin"



// IQ DC Offset For 11B in dB. Value is returned. DSSS/CCK only

#define WT_RES_FRAME_IQ_OFFSET_11B                    "iq.offset"



// RF Carrier Suppression for 11B in dB. Value is returned. DSSS/CCK only

#define WT_RES_FRAME_CARRIER_SUPPRESSION_11B    "carrier.suppression"



//Psdu Length In Frame. WIFI only

#define WT_RES_FRAME_PSDU_LENGTH                    "psdu.length"

//*****************************************************************************

//                                BT     Frame

//*****************************************************************************

//BT BR Freq.Drift,Hz(Payload 10101010)

#define WT_RES_BT_FRAME_CARR_FREQ_DRIFT              "BT_CARR_FREQ_DRIFT"



//BT BR Initial freq error Real vector, Hz
```

```
#define WT_RES_BT_FRAME_CARR_FREQ_BUF                "BT_CARR_FREQ_BUF"

//BT BR Max Initial freq offset of each burst detected in Hz.
#define WT_RES_BT_FRAME_MAX_CARR_FREQ
"BT_MAX_CARR_FREQ"

//BT BR Delta F1(and WT_ANA_PARM_BT_CARR_FREQ_DRIFT) valid
#define WT_RES_BT_FRAME_DELTA_F1_VALID               "BT_DELTA_F1_VALID"

//BT BR Delta F1 avg,Hz(Payload 00001111)
#define WT_RES_BT_FRAME_DELTA_F1_AVG                 "BT_DELTA_F1_AVG"

//BT BR Delta F2 valid
#define WT_RES_BT_FRAME_DELTA_F2_VALID               "BT_DELTA_F2_VALID"

//BT BR Delta F2 avg,Hz(Payload 10101010)
#define WT_RES_BT_FRAME_DELTA_F2_AVG                 "BT_DELTA_F2_AVG"

//BT BR Delta F2 max,Hz(Payload 10101010)
#define WT_RES_BT_FRAME_DELTA_F2_MAX                 "BT_DELTA_F2_MAX"

//BT EDR DEVM是否有效
```

```
#define WT_RES_BT_FRAME_BT_DEVM_VALID                "BT_DEVM_VALID"

//BT EDR DEVM,%
#define WT_RES_BT_FRAME_BT_DEVM                      "BT_DEVM"

//BT EDR DEVM Peak,%
#define WT_RES_BT_FRAME_BT_DEVM_PEAK                 "BT_DEVM_PEAK"

//BT EDR Power Diff,dB
#define WT_RES_BT_FRAME_BT_POW_DIFF                  "BT_POW_DIFF"

//BT EDR DEVM （BT 3M时小于20%, 2M时小于30%)的比例
#define WT_RES_BT_FRAME_BT_99PCT                     "BT_99PCT"

//BT EDR Omega I,Hz
#define WT_RES_BT_FRAME_EDR_Omega_I                  "BT_Omega_I"

//BT EDR Omega O,Hz
#define WT_RES_BT_FRAME_EDR_Omega_O                  "BT_Omega_O"

//BT EDR Omega IO,Hz
#define WT_RES_BT_FRAME_EDR_Omega_IO                 "BT_Omega_IO"
```

```
//BT Bandwidth-20dB Passed Or Failed. 1(1.0) - Passed; 0(0.0) - Failed.
#define WT_RES_BT_FRAME_BW20dB_Passed                    "BT_BW20dB_Passed"

//BT Bandwidth-20dB value
#define WT_RES_BT_FRAME_BW20dB                           "BT_BW20dB_Value"

//BT BLE FnMax,Hz
#define WT_RES_BT_FRAME_BLE_FnMax                        "BT_FnMax"

//BT BLE F0FnMax,Hz
#define WT_RES_BT_FRAME_BLE_F0FnMax                      "BT_F0FnMax"

//BT BLE Delta_F1F0,Hz
#define WT_RES_BT_FRAME_BLE_Delta_F1F0                   "BT_Delta_F1F0"

//BT BLE F0Fn5_Max,Hz
#define WT_RES_BT_FRAME_BLE_F0Fn5_Max                    "BT_F0Fn5_Max"

//*****************************************************************************
```

```
//                                  Zigbee     Frame
//*****************************************************************************
//Zigbee EVM(PSDU), dB
#define WT_RES_ZIGBEE_FRAME_EVM_PSDU                    "Zigbee.evm(psdu)"

//Zigbee EVM(PSDU), %
#define WT_RES_ZIGBEE_FRAME_EVM_PSDU_PERCENT
"Zigbee.evm(psdu).percent"

//Zigbee EVM(SHR+PHR), dB
#define WT_RES_ZIGBEE_FRAME_EVM_SHRPHR                  "Zigbee.evm(shr+phr)"

//Zigbee EVM(SHR+PHR), %
#define WT_RES_ZIGBEE_FRAME_EVM_SHRPHR_PERCENT
"Zigbee.evm(shr+phr).percent"
```

4.3 枚举定义

枚举定义主要包含VSA,VSG参数设置选项定义、错误码定义等。

4.3.1 VSG 状态定义

```
enum WT_VSG_STATE{
    WT_VSG_STATE_DONE,          //VSG结束
    WT_VSG_STATE_RUNNING,       //VSG正在发送信号
    WT_VSG_STATE_TIMEOUT,       //VSG超时
    WT_VSG_STATE_ERR_DONE,      //VSG错误
    WT_VSG_STATE_WAITING        //VSG等待
};
```

4.3.2 无线网络测试仪类型

```
enum WT_TESTER_TYPE
{
    WT160,
    WT200,
    WT208 = 3,
    WT400
};
```

4.3.3 错误码定义

```
enum WT_ERR_CODE_ENUM{

        WT_ERR_CODE_OK,

        //连接失败、未建立或已中断

        WT_ERR_CODE_CONNECT_FAIL,

        //参数有误

        WT_ERR_CODE_UNKNOW_PARAMETER,

        //内存不足

        WT_ERR_CODE_OUT_OF_MEMORY,

        //未抓取数据

        WT_ERR_CODE_NO_DATA_CAPTURED,

        //超时

        WT_ERR_CODE_TIMEOUT,

        //指定的VSG功率不准确

        WT_ERR_CODE_VsgInaccuracy,

        //其他错误

        WT_ERR_CODE_GENERAL_ERROR,

        //带宽设置错误
```

```
WT_ERR_CODE_BANDWIDTH_ERROR,

//信号类型错误

WT_ERR_CODE_SIGNALTYPE_ERROR,

//11n帧类型错误

WT_ERR_CODE_FRM_ERROR,

//参数关联错误

WT_ERR_CODE_PARAMETER_MISMATCH,

//PSDU设置错误

WT_ERR_CODE_PSDU_ERROR,

//Mac格式错误

WT_ERR_CODE_PSDU_CONVERT_FAIL,

//存放输出数据的指针错误

WT_ERR_CODE_OUTDATA_INVALID,

//生成信号失败

WT_ERR_CODE_GENERATE_FAIL,

//仪器没有指定的波形文件

WT_ERR_CODE_TESTER_NO_WAVE,

WT_ERR_CODE_LAST
```

```
};
```

4.3.4 测试结果数据类型

```
enum WT_RESULT_DATA_TYPE
{
    //1字节
    TYPE_int8,
    //2字节
    TYPE_int16,
    //4字节
    TYPE_int32,
    //8字节
    TYPE_double,
    //16字节
    TYPE_complex
};
```

4.3.5 Trigger 类型定义

```
enum WT_TRIG_TYPE_ENUM
{
    // Free running ADC sampling.
    WT_TRIG_TYPE_FREE_RUN,
    // ADC External Trigger selected.
    WT_TRIG_TYPE_EXT,
    // ADC IF Trigger selected - trigger calibration will be performed.
    WT_TRIG_TYPE_IF,
    WT_TRIG_TYPE_IF_NO_CAL
};
```

4.3.6 相位跟踪选项

```
enum WT_PH_CORR_ENUM
{
    // Phase correction off.
    WT_PH_CORR_OFF          = 1,
    // Symbol-by-symbol correction.   - Default value.
    WT_PH_CORR_SYM_BY_SYM = 2,
```

```
    // Moving avg. correction    - 10 symbols

    WT_PH_CORR_MOVING_AVG = 3

};
```

4.3.7 通道估计选项

```
enum WT_CH_EST_ENUM

{

    // Raw Channel Estimate - long train. - Default value.

    WT_CH_EST_RAW           = 1,

    // Same as WT_CH_EST_RAW.

    WT_CH_EST_RAW_LONG    = WT_CH_EST_RAW,

    // 2nd Order Polyfit.

    WT_CH_EST_2ND_ORDER = 2,

    // Raw Channel Estimate - full packet

    WT_CH_EST_RAW_FULL    = 3

};
```

4.3.8 时序跟踪选项

```
enum WT_SYM_TIM_ENUM

{

    // Symbol Timing Correction Off.

     WT_SYM_TIM_OFF = 1,

     // Symbol Timing Correction On - Default value.

     WT_SYM_TIM_ON   = 2

};
```

4.3.9 频率同步选项

```
enum WT_FREQ_SYNC_ENUM

{

     // Short Training Symbol.

     WT_FREQ_SYNC_SHORT_TRAIN = 1,

     // Long Training Symbol   - Default value.

     WT_FREQ_SYNC_LONG_TRAIN   = 2,

     // Full Data Packet.
```

```
    WT_FREQ_SYNC_FULL_PACKET = 3

};
```

4.3.10 幅度跟踪选项

```
enum WT_AMPL_TRACK_ENUM

{

    // Amplitude tracking off.   - Default value.

    WT_AMPL_TRACK_OFF = 1,

    // Amplitude tracking on

    WT_AMPL_TRACK_ON   = 2

};
```

4.3.11 直流去除选项

```
enum WT_DC_REMOVAL_ENUM

{
```

```
    // DC removal Off   - Default value.

    WT_DC_REMOVAL_OFF = 0,

    // DC removal On.

    WT_DC_REMOVAL_ON   = 1

};
```

4.3.12 11b 均衡类型选项

```
enum WT_EQ_ENUM

{

    // Equalizer Off. - Default value.

    WT_EQ_OFF      = 1,

    // 5 taps equalizer

    WT_EQ_5_TAPS = 5,

    // 7 taps equalizer.

    WT_EQ_7_TAPS = 7,

    // 9 taps equalizer.

    WT_EQ_9_TAPS = 9
```

```
};
```

4.3.13 11b 相位跟踪选项

```
enum WT_PH_CORR_11b_ENUM

{

    // Phase correction off.

     WT_PH_CORR_11b_OFF = 1,

     // Symbol-by-symbol correction. - Default value.

     WT_PH_CORR_11b_ON   = 2,

};
```

4.3.14 RF 口定义

```
enum WT_PORT_ENUM

{

     // Port has been disabled.

     WT_PORT_OFF = 1,
```

```
    // RF port 1
    WT_PORT_RF1 = 2,
    // RF port 2
    WT_PORT_RF2 = 3,
    // RF port 3, WT208 only
    WT_PORT_RF3 = 4,
    // RF port 4, WT208 only
    WT_PORT_RF4 = 5
};
```

4.3.15 IQ 交换(频谱反转) 选项

```
enum WT_IQ_SWAP_ENUM
{
    // IQ swap has been disabled.   -default
    WT_IQ_SWAP_DISABLED,
    // IQ swap has been enabled.
    WT_IQ_SWAP_ENABLED
};
```

4.3.16 信号解调类型定义

```
enum WT_DEMOD_ENUM
{
    // 11a/g
    WT_DEMOD_11AG,
    // 11b
    WT_DEMOD_11B,
    // 11n 20M
    WT_DEMOD_11N_20M,
    // 11n 40M
    WT_DEMOD_11N_40M,
    // 11ac 20M
    WT_DEMOD_11AC_20M,
    // 11ac 40M
    WT_DEMOD_11AC_40M,
    // 11ac 80M
```

```
    WT_DEMOD_11AC_80M,
    // 11ac 160M
    WT_DEMOD_11AC_160M,
    // 11ac 80+80M
    WT_DEMOD_11AC_80_80M,
    // Bluetooth
    WT_DEMOD_BT = 9,
    // Zigbee
    WT_DEMOD_ZIGBEE = 10,
    // Unknow
    WT_DEMOD_UNKNOW = 0xFF
};
```

4.3.17 仪器中预定义的信号类型

```
enum WT_SIGNAL_ENUM
{
    // Wave file Defined by user.
    SIG_USERFILE,
```

```
// Continuous Wave.
SIG_CW_SIN0,
// Continuous Wave Sin 100KHz.
SIG_CW_SIN100K,
// Continuous Wave Sin 1MHZ.
SIG_CW_SIN1M,
// Continuous Wave Sin -100KHZ.
SIG_CW_SINNEG100K,

// 802.11g   6 Mbit/s.
SIG_802_11_G_6MBS = 101,
// 802.11g   9 Mbit/s.
SIG_802_11_G_9MBS,
// 802.11g 12 Mbit/s.
SIG_802_11_G_12MBS,
// 802.11g 18 Mbit/s.
SIG_802_11_G_18MBS,
```

```
// 802.11g 24 Mbit/s.

SIG_802_11_G_24MBS,

// 802.11g 36 Mbit/s.

SIG_802_11_G_36MBS,

// 802.11g 48 Mbit/s.

SIG_802_11_G_48MBS,

// 802.11g 54 Mbit/s.

SIG_802_11_G_54MBS,



// 802.11n 20M MCS0.

SIG_HT_20_MCS0 = 201,

// 802.11n 20M MCS1.

SIG_HT_20_MCS1,

// 802.11n 20M MCS2.

SIG_HT_20_MCS2,

// 802.11n 20M MCS3.

SIG_HT_20_MCS3,

// 802.11n 20M MCS4.
```

```
SIG_HT_20_MCS4,

// 802.11n 20M MCS5.

SIG_HT_20_MCS5,

// 802.11n 20M MCS6.

SIG_HT_20_MCS6,

// 802.11n 20M MCS7.

SIG_HT_20_MCS7,



// 802.11n 40M MCS0.

SIG_HT_40_MCS0 = 301,

// 802.11n 40M MCS1.

SIG_HT_40_MCS1,

// 802.11n 40M MCS2.

SIG_HT_40_MCS2,

// 802.11n 40M MCS3.

SIG_HT_40_MCS3,

// 802.11n 40M MCS4.

SIG_HT_40_MCS4,
```

```
// 802.11n 40M MCS5.

SIG_HT_40_MCS5,

// 802.11n 40M MCS6.

SIG_HT_40_MCS6,

// 802.11n 40M MCS7.

SIG_HT_40_MCS7,



// 802.11ac 20M MCS0.

SIG_VHT_20_MCS0 = 401,

// 802.11ac 20M MCS1.

SIG_VHT_20_MCS1,

// 802.11ac 20M MCS2.

SIG_VHT_20_MCS2,

// 802.11ac 20M MCS3.

SIG_VHT_20_MCS3,

// 802.11ac 20M MCS4.

SIG_VHT_20_MCS4,

// 802.11ac 20M MCS5.
```

```
SIG_VHT_20_MCS5,

// 802.11ac 20M MCS6.

SIG_VHT_20_MCS6,

// 802.11ac 20M MCS7.

SIG_VHT_20_MCS7,

// 802.11ac 20M MCS8.

SIG_VHT_20_MCS8,



// 802.11ac 40M MCS0.

SIG_VHT_40_MCS0 = 501,

// 802.11ac 40M MCS1.

SIG_VHT_40_MCS1,

// 802.11ac 40M MCS2.

SIG_VHT_40_MCS2,

// 802.11ac 40M MCS3.

SIG_VHT_40_MCS3,

// 802.11ac 40M MCS4.

SIG_VHT_40_MCS4,
```

```
// 802.11ac 40M MCS5.

SIG_VHT_40_MCS5,

// 802.11ac 40M MCS6.

SIG_VHT_40_MCS6,

// 802.11ac 40M MCS7.

SIG_VHT_40_MCS7,

// 802.11ac 40M MCS8.

SIG_VHT_40_MCS8,

// 802.11ac 40M MCS9.

SIG_VHT_40_MCS9,



// 802.11ac 80M MCS0.

SIG_VHT_80_MCS0 = 601,

// 802.11ac 80M MCS1.

SIG_VHT_80_MCS1,

// 802.11ac 80M MCS2.

SIG_VHT_80_MCS2,

// 802.11ac 80M MCS3.
```

```
SIG_VHT_80_MCS3,

// 802.11ac 80M MCS4.

SIG_VHT_80_MCS4,

// 802.11ac 80M MCS5.

SIG_VHT_80_MCS5,

// 802.11ac 80M MCS6.

SIG_VHT_80_MCS6,

// 802.11ac 80M MCS7.

SIG_VHT_80_MCS7,

// 802.11ac 80M MCS8.

SIG_VHT_80_MCS8,

// 802.11ac 80M MCS9.

SIG_VHT_80_MCS9,



// 802.11b 1 Mbit/s.

SIG_802_11_B_1MBS = 701,

// 802.11b 2 Mbit/s.

SIG_802_11_B_2MBS,
```

```
// 802.11b 5.5 Mbit/s.

SIG_802_11_B_5_5MBS,

// 802.11b 11 Mbit/s.

SIG_802_11_B_11MBS,



//*** Begined with FW version 3.0.1.4 ***

// 802.11n 20M MCS0 ShortGI.

SIG_HT_20_MCS0_ShortGI = 1201,

// 802.11n 20M MCS1 ShortGI.

SIG_HT_20_MCS1_ShortGI,

// 802.11n 20M MCS2 ShortGI.

SIG_HT_20_MCS2_ShortGI,

// 802.11n 20M MCS3 ShortGI.

SIG_HT_20_MCS3_ShortGI,

// 802.11n 20M MCS4 ShortGI.

SIG_HT_20_MCS4_ShortGI,

// 802.11n 20M MCS5 ShortGI.

SIG_HT_20_MCS5_ShortGI,
```

```
// 802.11n 20M MCS6 ShortGI.

SIG_HT_20_MCS6_ShortGI,

// 802.11n 20M MCS7 ShortGI.

SIG_HT_20_MCS7_ShortGI,



// 802.11n 40M MCS0 ShortGI.

SIG_HT_40_MCS0_ShortGI = 1301,

// 802.11n 40M MCS1 ShortGI.

SIG_HT_40_MCS1_ShortGI,

// 802.11n 40M MCS2 ShortGI.

SIG_HT_40_MCS2_ShortGI,

// 802.11n 40M MCS3 ShortGI.

SIG_HT_40_MCS3_ShortGI,

// 802.11n 40M MCS4 ShortGI.

SIG_HT_40_MCS4_ShortGI,

// 802.11n 40M MCS5 ShortGI.

SIG_HT_40_MCS5_ShortGI,

// 802.11n 40M MCS6 ShortGI.
```

```
SIG_HT_40_MCS6_ShortGI,

// 802.11n 40M MCS7 ShortGI.

SIG_HT_40_MCS7_ShortGI,



// 802.11ac 20M MCS0 ShortGI.

SIG_VHT_20_MCS0_ShortGI = 1401,

// 802.11ac 20M MCS1 ShortGI.

SIG_VHT_20_MCS1_ShortGI,

// 802.11ac 20M MCS2 ShortGI.

SIG_VHT_20_MCS2_ShortGI,

// 802.11ac 20M MCS3 ShortGI.

SIG_VHT_20_MCS3_ShortGI,

// 802.11ac 20M MCS4 ShortGI.

SIG_VHT_20_MCS4_ShortGI,

// 802.11ac 20M MCS5 ShortGI.

SIG_VHT_20_MCS5_ShortGI,

// 802.11ac 20M MCS6 ShortGI.

SIG_VHT_20_MCS6_ShortGI,
```

```
// 802.11ac 20M MCS7 ShortGI.

SIG_VHT_20_MCS7_ShortGI,

// 802.11ac 20M MCS8 ShortGI.

SIG_VHT_20_MCS8_ShortGI,



// 802.11ac 40M MCS0 ShortGI.

SIG_VHT_40_MCS0_ShortGI = 1501,

// 802.11ac 40M MCS1 ShortGI.

SIG_VHT_40_MCS1_ShortGI,

// 802.11ac 40M MCS2 ShortGI.

SIG_VHT_40_MCS2_ShortGI,

// 802.11ac 40M MCS3 ShortGI.

SIG_VHT_40_MCS3_ShortGI,

// 802.11ac 40M MCS4 ShortGI.

SIG_VHT_40_MCS4_ShortGI,

// 802.11ac 40M MCS5 ShortGI.

SIG_VHT_40_MCS5_ShortGI,

// 802.11ac 40M MCS6 ShortGI.
```

```
SIG_VHT_40_MCS6_ShortGI,

// 802.11ac 40M MCS7 ShortGI.

SIG_VHT_40_MCS7_ShortGI,

// 802.11ac 40M MCS8 ShortGI.

SIG_VHT_40_MCS8_ShortGI,

// 802.11ac 40M MCS9 ShortGI.

SIG_VHT_40_MCS9_ShortGI,



// 802.11ac 80M MCS0 ShortGI.

SIG_VHT_80_MCS0_ShortGI = 1601,

// 802.11ac 80M MCS1 ShortGI.

SIG_VHT_80_MCS1_ShortGI,

// 802.11ac 80M MCS2 ShortGI.

SIG_VHT_80_MCS2_ShortGI,

// 802.11ac 80M MCS3 ShortGI.

SIG_VHT_80_MCS3_ShortGI,

// 802.11ac 80M MCS4 ShortGI.

SIG_VHT_80_MCS4_ShortGI,
```

```
// 802.11ac 80M MCS5 ShortGI.

SIG_VHT_80_MCS5_ShortGI,

// 802.11ac 80M MCS6 ShortGI.

SIG_VHT_80_MCS6_ShortGI,

// 802.11ac 80M MCS7 ShortGI.

SIG_VHT_80_MCS7_ShortGI,

// 802.11ac 80M MCS8 ShortGI.

SIG_VHT_80_MCS8_ShortGI,

// 802.11ac 80M MCS9 ShortGI.

SIG_VHT_80_MCS9_ShortGI,



// 802.11b 1 Mbit/s Short Preamble. Invalid.

SIG_802_11_B_1MBS_ShortPreamble = 1701,

// 802.11b 2 Mbit/s     Short Preamble.

SIG_802_11_B_2MBS_ShortPreamble,

// 802.11b 5.5 Mbit/s Short Preamble.

SIG_802_11_B_5_5MBS_ShortPreamble,

// 802.11b 11 Mbit/s    Short Preamble.
```

```
    SIG_802_11_B_11MBS_ShortPreamble,

    // Wave file Defined by Tester.

    SIG_TESTERFILE = 5000,

    SIG_LAST

};
```

4.3.18 带宽自动侦测

```
/*Enumeration for frame bandwidth dectecting.*/

enum WT_BANDWIDTH_DETECT_ENUM

{

    // - Default value.

    WT_USER_DEFINED,

    WT_BW_AUTO_DETECT,

    WT_AUTO_DETECT

};
```

4.3.19 信号调制源

```
  enum WT_SOURCE_ENUM

  {
```

```
    // Internal modulation from wave.
    WT_SOURCE_WAVE,
    // Internal modulation from signal generator.
    WT_SOURCE_SIGNAL_GENERATOR,
    // Undefined default modulation source.
    WT_SOURCE_UNDEFINED
  };
```

4.3.20 参考时钟

```
enum WT_REF_CLK_ENUM
{
    // Auto.
    WT_CLK_AUTO,
    // External Reference Clock.
    WT_CLK_EXT,
    // Internal Reference Clock.
    WT_CLK_INT
  };
```

4.3.21 BT DataRate

```
enum WT_BT_DATARATE

{

    WT_BT_DATARATE_Auto = 0,

    WT_BT_DATARATE_1M   = 1,

    WT_BT_DATARATE_2M   = 2,

    WT_BT_DATARATE_3M   = 3,

};
```

4.3.22 BT 包类型

```
enum WT_BT_PACKETTYPE

{

    WT_BT_PACKETTYPE_NULL   = 0,

    WT_BT_PACKETTYPE_POLL   = 1,

    WT_BT_PACKETTYPE_FHS    = 2,

    WT_BT_PACKETTYPE_DH1    = 3,

    WT_BT_PACKETTYPE_DH3    = 4,

    WT_BT_PACKETTYPE_DH5    = 5,

    WT_BT_PACKETTYPE_DM1    = 6,

    WT_BT_PACKETTYPE_DM3    = 7,
```

```
    WT_BT_PACKETTYPE_DM5     = 8,
    WT_BT_PACKETTYPE_HV1     = 9,
    WT_BT_PACKETTYPE_HV2     = 10,
    WT_BT_PACKETTYPE_HV3     = 11,
    WT_BT_PACKETTYPE_DV      = 12,
    WT_BT_PACKETTYPE_AUX1    = 13,
    WT_BT_PACKETTYPE_EV3     = 14,
    WT_BT_PACKETTYPE_EV4     = 15,
    WT_BT_PACKETTYPE_EV5     = 16,
    WT_BT_PACKETTYPE_2_DH1 = 17,
    WT_BT_PACKETTYPE_2_DH3 = 18,
    WT_BT_PACKETTYPE_2_DH5 = 19,
    WT_BT_PACKETTYPE_3_DH1 = 20,
    WT_BT_PACKETTYPE_3_DH3 = 21,
    WT_BT_PACKETTYPE_3_DH5 = 22,
    WT_BT_PACKETTYPE_2_EV3 = 23,
    WT_BT_PACKETTYPE_2_EV5 = 24,
    WT_BT_PACKETTYPE_3_EV3 = 25,
    WT_BT_PACKETTYPE_3_EV5 = 26
};
```

4.3.23 VSG 波形文件操作

```
enum TERSTER_WAVE_OPERATE_OPTION

{

    //添加信号数据

    ADD_WAVE,

    //删除信号

    REMOVE_WAVE

};
```

4.3.24 VSA 重采样配置

```
enum TESTER_SAMPLERATE_MODE

{

    //采样率默认模式,采样率设为 120M

    RATE_DEFAULT,

    //采样率自动匹配模式,采样率根据标准和带宽匹配最佳采样率

    RATE_AUTO

};
```

4.3.25 EVM 符号类型

```
enum SYMBOLS_EVM_RESULT_TYPE
{
    //取平均值
    AVERAGE_VALUE,
    //取最大值
    MAX_VALUE,
    //取最小值
    MIN_VALUE
};
```

5 Demo 指导

以VC为例	
步骤1	使用WLAN.Tester.API时，应用程序工程的“附加库目录”、“附加资源目录”设置为WLAN.Tester.API相关文件所在的文件夹
步骤2	在应用程序工程中调用WLAN.Tester.API.dll并包含头文件tester.h 示例如下： #include "tester.h" #pragma comment(lib, "WLAN.Tester.API.lib")

备注	如果调用该Dll的windows环境下的C程序，还需在#include "tester.h" 之前添加如下定义：#define　WT_C 例子中的m_vsgParameter，m_vsaParameter为仪器对象中的成员

5.1 DLL 初始化以及释放

初始化DLL资源	static void Initialize()
	{
	　　//调用WT_DLLInitialize()来初始化DLL环境,如果不初始化就直
	　　//接使用其他接口，可能导致返回的错误码为
	　　//WT_ERR_CODE_UNKNOW_PARAMETER
	　　WT_DLLInitialize();
	　}

释放DLL资源	static void Terminate()
	{
	　　//在完成对WLAN.Tester.API.dll调用之后需释放该DLL所占资源
	　　WT_DLLTerminate();
	　}

备注	在使用WLAN.Tester.API之前，需“初始化该DLL资源”的各种处理机 制，并且在停止使用该DLL时，需“释放该DLL资源”

5.2 连接与断开

连接仪器	bool TesterControler::ConnentTester(char *ip)
	{
	int connID = -1;
	//连接指定的WT无线网络测试仪
	int result = **WT_Connect(ip, &connID)**;
	if (result == WT_ERR_CODE_OK)
	{
	//连接成功后，保存连接标识ID,可以设置所连接到
	//WT无线网络测试仪与DUT之间的外部增益
	//保存连接ID
	m_connID = s32connID;
	//设置所连接到WT无线网络测试仪与DUT之间的外部增益
	WT_SetExternalGain(connID,-1.0);
	…
	return true;
	}
	else
	{
	return false;
	}

	}

断开仪器	bool TesterControler::DisConnectTester()
	{
	//断开连接
	WT_DisConnect(m_connID);
	return true;
	}

备注	在乒乓模式下，连接断开需每次测试都需要进行；独占端口模式则无需频繁连接断开。WT208多用户连接模式下，最多可连接4个用户

5.3 参数获取与设置

5.3.1 获取默认配置参数

获取默认配置参数	//建议在对WT无线网络测试仪进行设置之前
	//获取默认的VSA,VSG参数
	WT_GetDefaultParameter(&m_vsaParameter,
	&m_vsgParameter);

5.3.2 VSG 参数设置

```
int retResult = WT_ERR_CODE_OK;

//指定信号文件源是本地文件

m_vsgParameter.waveType = SIG_USERFILE;

//指定信号文件路径时需指定绝对路径

m_vsgParameter.wave = "E:\\wave\\54 Mbps(OFDM).csv";

//指定信号发送频率

m_vsgParameter.freq = 2412 * 1e6;

//VSG发送次数

m_vsgParameter.repeat = 10000;

//帧间隔，50us

m_vsgParameter.wave_gap = 50;

//tx功率,dBm

m_vsgParameter.power = -50;

//设置tx参数

result = WT_SetVSG(m_connID, &m_vsgParameter);



if (result == WT_ERR_CODE_UNKNOW_PARAMETER)

{

    //指定的波形文件有误

}
```

```
//其余判断略
```

5.3.3 VSA 参数设置

```
//可根据具体情况对VSA参数进行调整后再进行设置
//参考电平设置部分，在频偏校准等DUT发送功率未知时
//尽量设置为DUT大致功率 + 18dB，
//避免设置参考电平过低导致信号严重失真
//在验证阶段，则将该值设置为DUT功率 + 14dB
//如果输入功率范围未知,建议使用WT_SetVSA_AutoRange
m_vsaParameter.max_power = 30;
//指定接收频率
m_vsaParameter.freq = 2412 * 1e6;
//指定信号抓取长度，采集长度
m_vsaParameter.smp_time = 2000;
//分析模式,见 WT_DEMOD_ENUM
m_vsaParameter.demod = WT_DEMOD_11AG;
//触发类型,见 WT_TRIG_TYPE_ENUM
m_vsaParameter.trig_type = WT_TRIG_TYPE_IF;
//触发超时时间
 m_vsaParameter.trig_timeout = 20;
```

```
//设置rx参数
result = WT_SetVSA(m_connID, &m_vsaParameter);
//判断略
…
//根据指定rx参数，自动调整max_power值设置rx
result = WT_SetVSA_AutoRange(m_connID, &m_vsaParameter);
```

5.3.4 WT-208 子网口参数设置

子网口参数配置为WT-208特有的接口，其主要作用是解决在同一子网内多个相同IP的DUT IP冲突问题，可以使用同一PC控制相同IP的多个DUT进行测试，多个EXE配置同一WT-208仪器时，如果配置项相同，不会影响其他子网口的正常通讯；

注意点：	
1	如果不需要启动WT-208的子网口，则无需操作该接口
2	PC as TFTP server与DUT as TFTP server不能同时存在， 具体使用请参考APIsample中的相关例子。
3	DUT实际IP与仪器IP不能在同一子网内 例如仪器的IP为192.168.10.254，掩码为255.255.255.0时，DUT的IP不能为192.168.10.x

配置例子：

```
/*****************************************
```

```
* 基础信息

*****************************************/

//4个子网口虚拟IP

char s8VirAddr[4][16] =

{

    {"192.168.100.11"},

    {"192.168.100.12"},

    {"192.168.100.13"},

    {"192.168.100.14"}

};

//DUT实际工作时的IP

char s8DutAddr[16] = {"192.168.1.6"};



/****   DUT作为TFTP服务器时的  Server IP ****/

char s8DutTftpServerAddr[16] = {"192.168.1.100"};



/**** PC作为TFTP服务器时的配置部分  ****/

// DUT指定的TFTP Server IP

char s8TftpServerAddr[16] = {"192.168.2.100"};

// DUT在TFTP时指定的Client IP

char s8TftpClientAddr[16] = {"192.168.2.1"};

// PC作为Server的实际IP
```

```
char s8TftpServerPCAddr[16] = {"192.168.100.100"};
//子网口IP结构
WT_208_NET_TYPE netTypeHdl = {0};
//基础部分赋值，必须赋值
strcpy_s(netTypeHdl.VirAddr1, s8VirAddr[0]);
strcpy_s(netTypeHdl.VirAddr2, s8VirAddr[1]);
strcpy_s(netTypeHdl.VirAddr3, s8VirAddr[2]);
strcpy_s(netTypeHdl.VirAddr4, s8VirAddr[3]);
strcpy_s(netTypeHdl.DutAddr, s8DutAddr);

if(DUT as TFTP server)
{
    //-----DUT as TFTP server----
    strcpy_s(netTypeHdl.DutTftpServerAddr, s8DutTftpServerAddr);
}
If(PC as TFTP server)
{
    //----PC as TFTP server；该部分与DUT as TFTP server不能同时存在----
    strcpy_s(netTypeHdl.TftpServerAddr, s8TftpServerAddr);
    strcpy_s(netTypeHdl.TftpClientAddr, s8TftpClientAddr);
    strcpy_s(netTypeHdl.TftpServerPCAddr, s8TftpServerPCAddr);
}
```

```
//赋值完成后，配置WT208子网口信息到仪器
result = WT_SetWT208NetInfo(m_connID, &netTypeHdl);
//判断略
```

5.4 信号抓取与分析

```
int retResult = WT_ERR_CODE_OK;
//根据指定的rx参数，抓取WLAN数据
retResult = WT_DataCapture(m_connID);
if (retResult == WT_ERR_CODE_OK)
{
    double anaResult = 0;
    char description[100] = {0};
    char unit[10] = {0};

    //抓取WLAN数据成功，可以对抓取到的数据做多种结果分析
    retResult = WT_GetResult(
                    m_connID ,
                    WT_RES_POWER_FRAME_DB,
                    &anaResult,
```

```
                    description,
                    unit);
if (retResult == WT_ERR_CODE_OK)
{
    printf("分析%s成功，值为%1.2f%s\r\n", description, anaResult, unit);
}
else
{
    printf("分析Power失败!\r\n");
    return false;
}
//以下部分判断略
retResult = WT_GetResult(
                   m_connID ,
                    WT_RES_FRAME_EVM_ALL,
                    &anaResult,
                    description,
                    unit);
retResult = WT_ GetResult(
                    m_connID ,
                    WT_RES_FRAME_EVM_PEAK,
                    &anaResult,
```

```
                    description,
                    unit);
    retResult = WT_ GetResult(
                    m_connID ,
                    WT_RES_FRAME_FREQ_ERR,
                    &anaResult,
                    description,
                    unit);
        …
    return true;
}
else
{
    //其余判断略
    return false;
}
```

注意：

在获取帧功率(WT_RES_POWER_FRAME_DB)时 ,如果返回值正常 ,但功率值为-999.9(不包含外部衰减)；表明功率分析成功但无帧功率。例如：在连续波模式下就不会有帧功率。

获取BT结果时，需注意各结果项的有效性判断。

5.5 信号发送与停止

```
int result = WT_ERR_CODE_OK;
//可以对VSG参数进行调整
//如果指定的RF口之前为WT无线网络测试仪VSA所用，
//必须调用SetVSG将该接口切换至VSG
WT_SetVSG(&m_vsgParameter);

//以下部分为同步发送示例，常规情况下会使用同步发送
result = WT_StartVSG(m_connID);
if (result == WT_ERR_CODE_OK)
{
    //发送完成
    return true;
}
else
{
    //发送失败
    return false;
}
```

```
//以下部分为异步发送示例，BT测试时，使用异步发送的情况比较多
result = WT_AsynStartVSG (m_connID);
if (result == WT_ERR_CODE_OK)
{
    //开启异步发送成功
    return true;
}
else
{
    //开启异步发送失败
    return false;
}

bool TesterControler::StopVSG()
{
    //停止发送
    return WT_StopVSG(m_connID) == WT_ERR_CODE_OK;
}

Boolean TesterController::GetVsgState(int *state)
{
    s32 result = WT_ERR_CODE_OK;
```

```
//在异步发送过程中，或者多线程操作同一VSG过程中，
//可能需要获取当前状态来判断VSG是否已完成
    result = WT_GetVSGCurrentState(m_connID, state);
    return (result == WT_ERR_CODE_OK);
}
```

5.6 VSG 波形文件下载和使用

注意：

- WT-208 固件版本低于 3.0.1.13 不支持该项。
- WT200 固件版本低于 2.2.1.27 的不支持该项
- WT160 不支持该项
- 提供了 1GB 的存储空间给自定义信号文件

```
const char *waveName = "54 Mbps(OFDM).csv";
const char *localwaveform = "D:\\wave\\54 Mbps(OFDM).csv";
//把本地信号文件下载到仪器内部
```

```
//下载完成后，以后可以仪器内部保存的信号文件
int DownLoadWaveform(int connect_id, int forceDownload)
{
    int result = WT_ERR_CODE_OK;
    int testerExistWave = 0;
    if(forceDownload)
    {
        //强制更新。此时不必查询仪器内部是否存在该信号文件，强制覆盖
        result = WT_OperateTesterWave(connect_id,localwaveform,ADD_WAVE);
    }
    else
    {
        //查询信号文件在仪器内部是否存在
        result = WT_QueryTesterWave(connect_id,waveName,&testerExistWave);
        if(WT_ERR_CODE_OK == result)
        {
            //仪器内部不存在名字为waveName的信号文件
            if (0 == testerExistWave)
            {
                //把路径为localwaveform的信号文件下载到仪器内部
                result =
WT_OperateTesterWave(connect_id,localwaveform,ADD_WAVE);
```

```
            }
        }
    }
    return result;
}

int SetVSGParm(int connect_id)
{
    int result = WT_ERR_CODE_OK;
    VsgParameter m_vsgParameter;
    //使用下载信号文件到仪器内部
    if(WT_ERR_CODE_OK == DownLoadWaveform(connect_id,0))
    {
        //初始化m_vsgParameter参数，详细情况查看5.3.2
        //注意下面的波形文件类型和波形文件的赋值
        .....
        //指定信号文件类型,SIG_TESTERFILE:使用仪器存储的wave文件
        m_vsgParameter.waveType = SIG_TESTERFILE;
        //指定信号文件名字，非文件的绝对路径
        m_vsgParameter.wave = waveName;
        //设置tx参数
        result = WT_SetVSG(m_connID, &m_vsgParameter);
```

```
if (result == WT_ERR_CODE_UNKNOW_PARAMETER)
{
    //指定的波形文件有误
}
//如果指定的RF口之前为WT无线网络测试仪VSA所用，
//必须调用SetVSG将该接口切换至VSG
WT_SetVSG(&m_vsgParameter);
//以下部分为同步发送示例，常规情况下会使用同步发送
result = WT_StartVSG(m_connID);
if (result == WT_ERR_CODE_OK)
{
    //发送完成
    return true;
}
else
{
   //发送失败
   return false;
}

//以下部分为异步发送示例，BT测试时，使用异步发送的情况比较多
result = WT_AsynStartVSG (m_connID);
```

```
        if (result == WT_ERR_CODE_OK)
        {
            //开启异步发送成功
            return true;
        }
        else
        {
             //开启异步发送失败
             return false;
        }
    }
}

bool TesterControler::StopVSG()
{
    //停止发送
    return WT_StopVSG(m_connID) == WT_ERR_CODE_OK;
}

Boolean TesterController::GetVsgState(int *state)
{
    s32 result = WT_ERR_CODE_OK;
```

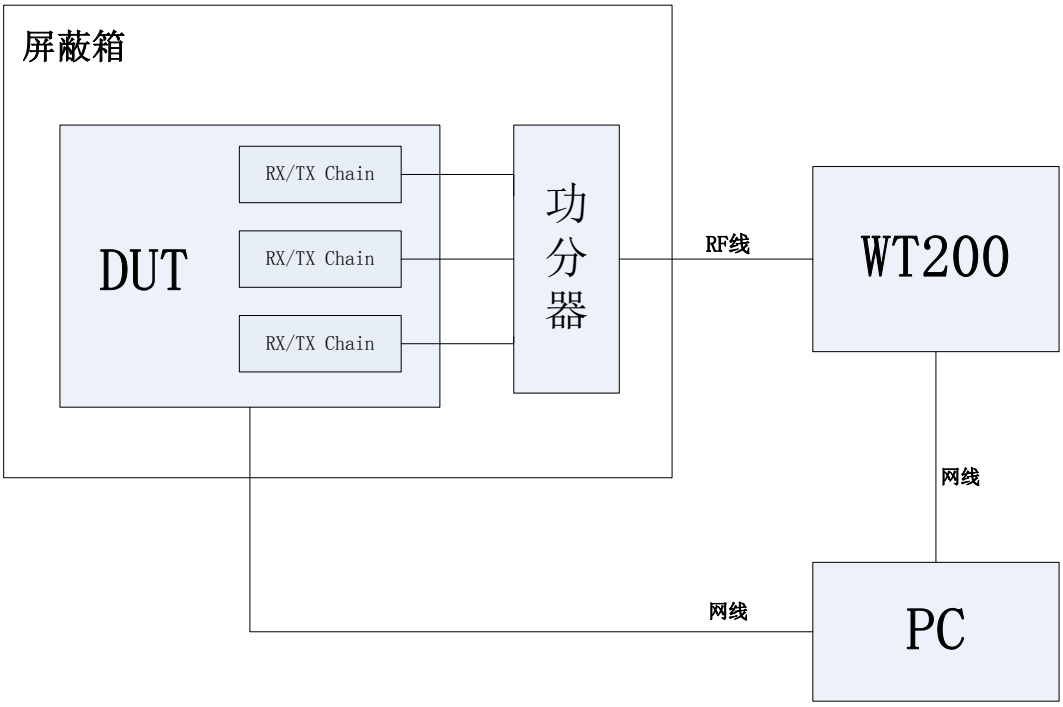
```
    //在异步发送过程中，或者多线程操作同一VSG过程中，
    //可能需要获取当前状态来判断VSG是否已完成
    result = WT_GetVSGCurrentState(m_connID, state);
    return (result == WT_ERR_CODE_OK);
}
```

5.7 BT VSA 参数配置

测试 BT EDR 数据时，建议 Trigger 前置时间跳过 GFSK 段，设置为 150us。而 WIFI 使用的 trigger 前置时间默认为 20us

5.8 Beamforming 使用

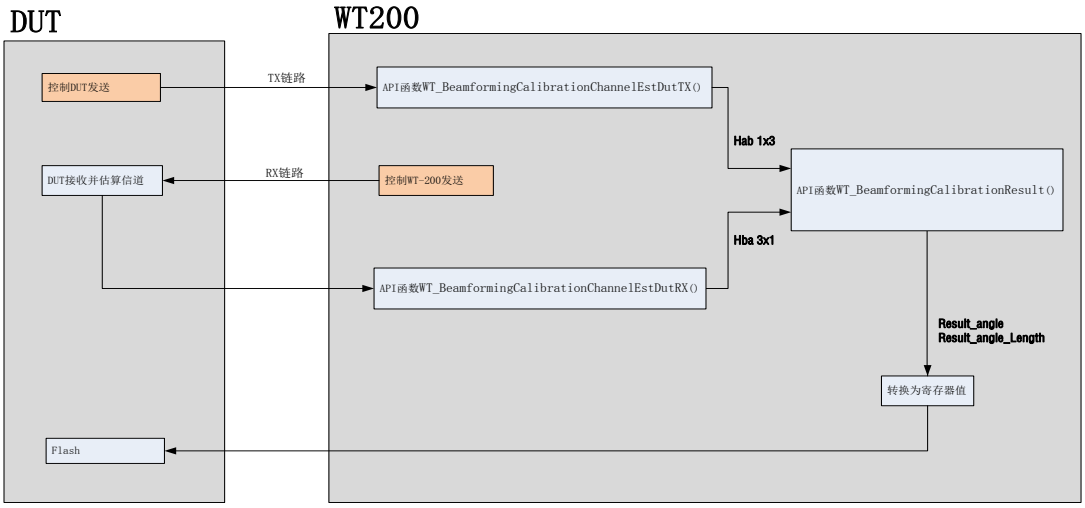
5.8.1 测试组网设置



说明：将 DUT 和功分器放置在屏蔽箱中，把 DUT 的多根天线连接到功分器上。

5.8.2 校准过程

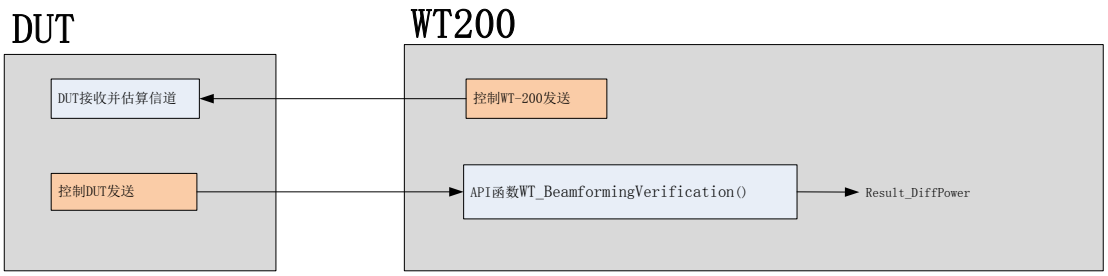
原理图



测试步骤：	
步骤 1	DUT 发送 MIMO 数据，WT-200 接收数据，由 WT-200 计算信道估计 Hab 1x3
步骤 2	WT-200 发送 SISO 数据，DUT 接收数据，由 DUT 计算信道估计 Hba 3x1
步骤 3	用步骤 1 的信道估计 Hab 1x3 和步骤 2 的信道估计结果 Hba 3x1，计算各天线的相位值
步骤 4	把各天线的相位值结果写入 DUT 的存储中

5.8.3 验证过程

原理图

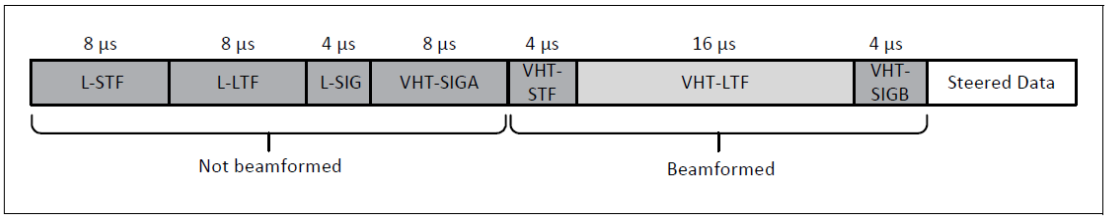


测试步骤：	
步骤1	控制DUT读取存储中Beamforming校准结果。
步骤2	配置DUT有关Beamforming寄存器。

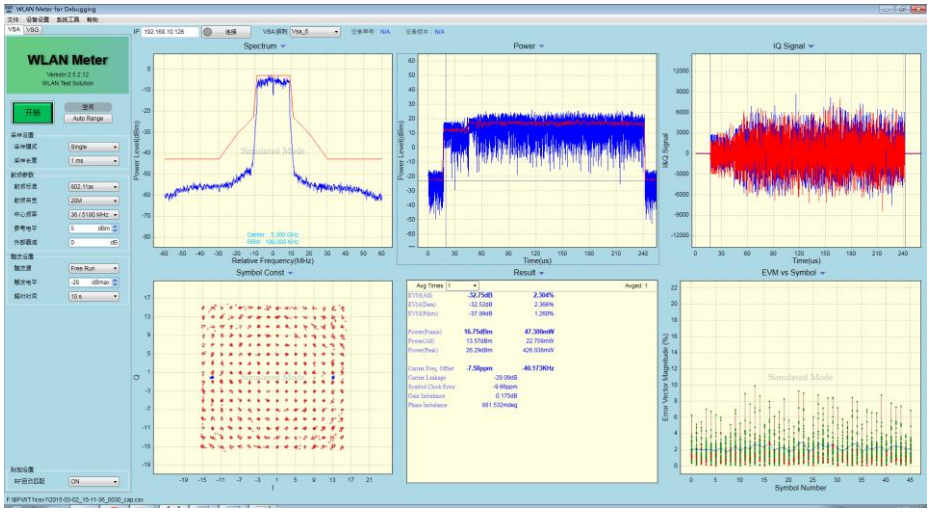
步骤3	WT-200发送SISO数据给DUT， 数据内容为VHT、MCS0、Nss1、RSSI:–50dBm； 由DUT内部估算当前信道信息。
步骤4	DUT发送Beamforming数据，WT-200计算出由Beamforming带来的增益值；判断功率增益值是否达到预期值。

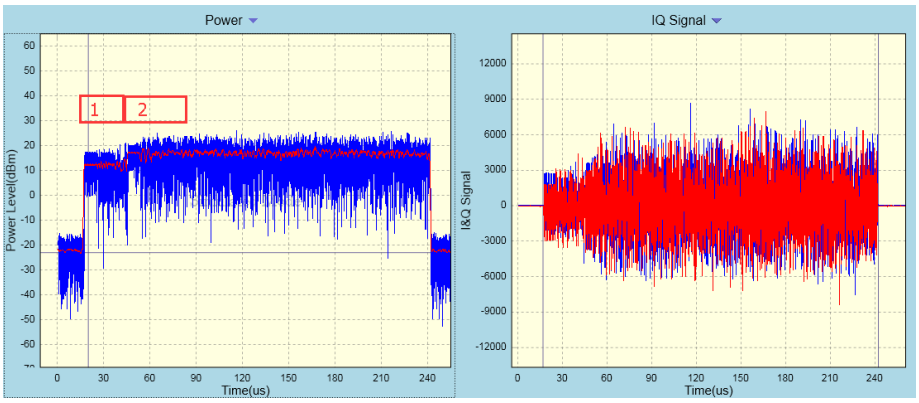
5.8.4 验证图解

下面解释在 Verification 时，Beamforming 所带来的增益测量。

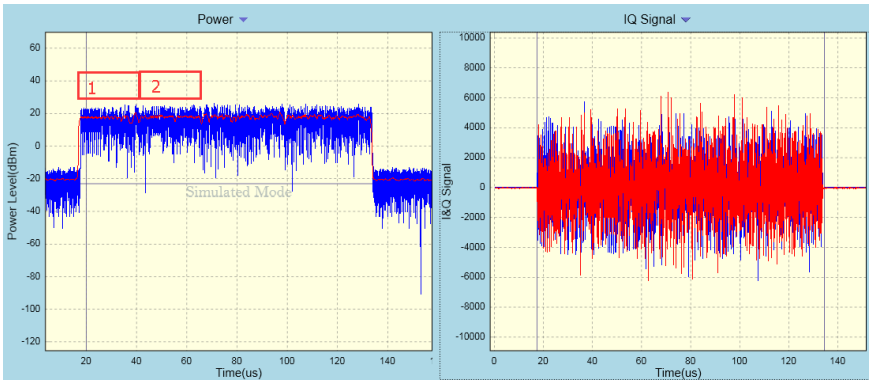


BCM4360 在做 Verification 时，在 legacy OFDM 部分和 VHT-SigA 都没有使用 Beamforming，但在后 VHT 部分都使用了；所以在测量时，是通过对比 legacy 部分和 VHT 部分功率差异来确定 Beamforming 所带来的增益





Beamforming 开启状态输出信号



Beamforming 关闭状态输出信号

从两个图形可以很明显的看出 Beamforming 使用后信号功率增强

5.8.5 Beamforming 示例

结合 WT API 说明，Demo 代码如下

```
#include "stdafx.h"

#include "tester.h"

#include <windows.h>

#pragma comment(lib,"WLAN.Tester.API.lib")
```

```
//通道估计数组容量

#define pcal_max_size 8192

//beamforming  目标增益

#define BFGAIN_TARGET 3

#define max_state -99999999

#define ant_num 3

//BF  校准

int BF_Cal();

//BF  测试

int BF_Verify();

//DUT  设置

int Dut_Setup(int freq,int demod,int datarate,double txpower){return 0;};

//DUT 发送 3 个 stream 的信号

int Dut_TxFrame_1s(){return 0;};

//    DUT 发送 1 个 stream 的信号

int Dut_TxFrame_3s(){return 0;};

//停止发送信号

int Dut_TxStop(){return 0;};



//使 Dut 进入接收状态

int Dut_RxReady(){return 0;};
```

```
//获取通道估计值
int Dut_ObtainChannelEst(int *phych4rpcal){return 0;};
//设置相位差
int Dut_SetRpcalvars(int freq,double *data){return 0;};
//写入相位差
int Dut_WriteRpcalvars(){return 0;};
VsaParameter *pVsaPa;
VsgParameter *pVsgPa;
//WT_HNDL_EX stHndlEx;
int connID;
int main(int argc, char* argv[])
{
    pVsaPa=new VsaParameter;
    pVsgPa=new VsgParameter;
    WT_GetDefaultParameter(pVsaPa,pVsgPa);
    int errcode;
    char ipAddr[100]="192.168.10.254";
    char buffer[100]={0};
    errcode=WT_Connect(ipAddr,&connID);
    if (errcode!=WT_ERR_CODE_OK)
    {
        WT_GetTesterConnStataus(ipAddr,buffer,strlen(buffer));
```

```
        printf("%s,connect tester fail\n",buffer);
        return -1;
    }
    //rf 参数
    pVsaPa->rfPort = WT_PORT_RF1;
    pVsaPa->smp_time = 2000;
    pVsaPa->trig_type = WT_TRIG_TYPE_IF;
    pVsaPa->trig_level = -31;
    pVsaPa->trig_timeout = 20;
    pVsaPa->trig_pretime = 20e-6;
    pVsaPa->demod = WT_DEMOD_11AC_20M;
    pVsgPa->rfPort= WT_PORT_RF1;
    pVsgPa->wave_gap=50;
    pVsgPa->waveType=SIG_VHT_20_MCS0;
    printf("BF Calibration start\n");
    if (BF_Cal()!=0)
    {
        printf("BF Calibration fail\n");
        return -1;
    }
    printf("BF Calibration end \n");
    printf("BF Verification start\n");
```

```
    if (BF_Verify()!=0)
    {
        printf("BF Verification fail");
        return -1;
    }
    printf("BF Verification end\n");
    printf("Write BF Rpcalvars\n");
    if (Dut_WriteRpcalvars()!=0)
    {
        return -1;
    }
    printf("==================Test End====================\n");
    delete pVsaPa;
    delete pVsgPa;
    return 0;
}
#define freq_num 4
int BF_Cal()
{
    //通道分组：
    int freq_cal[freq_num]={5200,5500,5560,5765};
    for (int freq_index=0;freq_index<freq_num;freq_index++)
```

```
{
    //====================H_AB=======================
    //配置 DUT,使用 VHT20 校准
    if(Dut_Setup(freq_cal[freq_index],
        WT_DEMOD_11AC_20M,SIG_VHT_20_MCS0,15)!=0)
    {
        return -1;
    }
    //发送 3 个 stream 信号
    if (Dut_TxFrame_3s()!=0)
    {
        return -1;
    }
    pVsaPa->freq = freq_cal[freq_num] * 1e6 ;
    //仪器 RX 设置
    if (WT_SetVSA(connID,pVsaPa)!=WT_ERR_CODE_OK)
    {
        return -1;
    }
    //仪器抓取
    if (WT_DataCapture(connID)!=WT_ERR_CODE_OK)
    {
```

```
        return -1;
    }
    //估算出 Hab 1x3
    if (WT_BeamformingCalibrationChannelEstDutTX(
        connID,
        WT_DEMOD_11AC_20M)!=WT_ERR_CODE_OK)
    {
        return -1;
    }
    //DUT 停止发送信号
    if (Dut_TxStop()!=0)
    {
        return -1;
    }
    //==================H_BA====================
    //配置 DUT,使用 VHT20 校准
    if(Dut_Setup(
        freq_cal[freq_index],
        WT_DEMOD_11AC_20M,
        SIG_VHT_20_MCS0,15)!=0)
    {
        return -1;
```

```
}
pVsgPa->freq = freq_cal[freq_num] * 1e6 ;
pVsgPa->power=-50;
pVsgPa->repeat=0;
//pVsgPa->rfPort
//仪器 TX 设置
if (WT_SetVSG(connID,pVsgPa)!=WT_ERR_CODE_OK)
{
    return -1;
}
//仪器发送 1 个 stream 的信号
if (WT_StartVSG(connID)!=WT_ERR_CODE_OK)
{
    return -1;
}
//DUT 进入接收状态
if (Dut_RxReady()!=0)
{
    return -1;
}
//计时
Sleep(1000);
```

```
//DUT 获取通道估计值
int phych4rpcal[pcal_max_size]={max_state};
if (Dut_ObtainChannelEst(phych4rpcal)!=0)
{
    return -1;
}
int count=0;
for (count=0;count<pcal_max_size;count++)
{
    if (phych4rpcal[count]=max_state)
    {
        break;
    }
}

//估算出 Hba 3x1
if (WT_BeamformingCalibrationChannelEstDutRX(
        connID,
        (double*)phych4rpcal,
        count)!=WT_ERR_CODE_OK)
{
    return -1;
```

```
}
double Result_angle[8]={0};
int angle_len;
if(WT_BeamformingCalibrationResult(
    connID,
    Result_angle,
    &angle_len)!=WT_ERR_CODE_OK)
//停止仪器发送信号
if (WT_StopVSG(connID)!=WT_ERR_CODE_OK)
{
    return -1;
}
double data[ant_num]={0};
for (count=0;count<ant_num-1;count++)
{
    data[count]=Result_angle[count];
}
//DUT 临时设置相位差
if (Dut_SetRpcalvars(freq_cal[freq_index],data)!=0)
{
    return -1;
}
```

```
    }
    return 0;
}
int BF_Verify()
{
    //通道分组：
    int freq_ver[freq_num]={5200,5500,5560,5765};
    for (int freq_index=0;freq_index<freq_num;freq_index++)
    {
        //配置 DUT,使用 VHT20 测试
        if(Dut_Setup(
                freq_ver[freq_index],
                WT_DEMOD_11AC_20M,
                SIG_VHT_20_MCS0,15)!=0)
        {
            return -1;
        }
        pVsgPa->freq = freq_ver[freq_num] * 1e6 ;
        pVsgPa->power=-50;
        pVsgPa->repeat=0;
```

```
//仪器 TX 设置


if (WT_SetVSG(connID,pVsgPa)!=WT_ERR_CODE_OK)
{
    return -1;
}
//仪器发送 1 个 stream 的信号
if (WT_StartVSG(connID)!=WT_ERR_CODE_OK)
{
    return -1;
}
//DUT  进入接收状态
if (Dut_RxReady()!=0)
{
    return -1;
}
//计时
Sleep(1000);
//配置 DUT,使用 VHT20 测试
if(Dut_Setup(
    freq_ver[freq_index],
    WT_DEMOD_11AC_20M,
```

```
        SIG_VHT_20_MCS0,15)!=0)
    {
        return -1;
    }
    //发送 1 个 stream 信号
    if (Dut_TxFrame_1s()!=0)
    {
        return -1;
    }
    if (WT_StopVSG(connID)!=WT_ERR_CODE_OK)
    {
        return -1;
    }
    pVsaPa->freq = freq_ver[freq_num] * 1e6 ;
    //仪器 RX 设置
    if (WT_SetVSA(connID,pVsaPa)!=WT_ERR_CODE_OK)
    {
        return -1;
    }
    //仪器抓取
    if (WT_DataCapture(connID)!=WT_ERR_CODE_OK)
    {
```

```
            return -1;
        }

        //DUT 停止发送信号
        if (Dut_TxStop()!=0)
        {
            return -1;
        }
        double dGain;
        //仪器计算 beamforming 增益
        if (WT_BeamformingVerification(connID,&dGain)!=WT_ERR_CODE_OK)
        {
            return -1;
        }
        if (dGain<BFGAIN_TARGET)
        {
            printf("frequency %d beamforming didn't reach our target:%ddb",
                    freq_ver[freq_index],
                    BFGAIN_TARGET);
            return -1;
        }
    }
```

```
    return 0;

}
```

5.9 VSA 参数与 IQxel-M 函数对应表

- **802.11ag：**
  - IQ 函数：LP_Analyze80211ag(int ph_corr_mode,
    int ch_estimate,
    int sym_tim_corr,
    int freq_sync,
    int ampl_track,
    double prePowStartSec,
    double prePowStopSec);
  - API VsaParameter(*4.1.1*)对应参数：

    ph_corr_mode<-->ph_corr

    ch_estimate<-->ch_estimate

    sym_tim_corr<-->sym_tim_corr

    freq_sync<-->freq_sync

    ampl_track<-->ampl_track
- **802.11b:**
  - IQ 函数： LP_Analyze80211b(int eq_taps,
    int DCremove11b_flag,

int method_11b,

double prePowStartSec,

double prePowStopSec);

- API VsaParameter(*4.1.1*)对应参数：

  eq_taps<-->eq_taps,

  DCremove11b_flag<-->dc_removal

- **802.11n：**

  - IQ 函数: int LP_Analyze80211n(char *type,

    char *mode,

    int enablePhaseCorr,

    int enableSymTimingCorr,

    int enableAmplitudeTracking,

    int decodePSDU,

    int enableFullPacketChannelEst,

    char *referenceFile,

    int packetFormat,

    int frequencyCorr,

    double prePowStartSec,

    double prePowStopSec);

  - API VsaParameter(*4.1.1*)对应参数：

    enablePhaseCorr<-->ph_corr

    enableSymTimingCorr<-->sym_tim_corr

enableAmplitudeTracking<-->ampl_track

enableFullPacketChannelEst<-->ch_estimate,类型 WT_CH_EST_ENUM

frequencyCorr<-->freq_sync

- **802.11AC：**
  - IQ 函数：LP_Analyze80211ac(char *mode ="nxn",

int enablePhaseCorr,

int enableSymTimingCorr,

int enableAmplitudeTracking,

int decodePSDU,

int enableFullPacketChannelEst,

int frequencyCorr,

char *referenceFile,

int packetFormat,

int numberOfPacketToAnalysis,

double prePowStartSec,

double prePowStopSec);

  - API VsaParameter(*4.1.1*)对应参数：

enablePhaseCorr<-->ph_corr

enableSymTimingCorr<-->sym_tim_corr

enableAmplitudeTracking<-->ampl_track

enableFullPacketChannelEst<-->ch_estimate,类型 WT_CH_EST_ENUM

frequencyCorr<-->freq_sync

6 使用指引

本章以添加WLAN产品的生产测试程序为例，介绍WLAN.Tester.API的使用方式，主要包含初始化/释放该程序、连接/断开仪器、频偏校准、功率校准、TX Verify、RX Verify等几个方面进行介绍。

本章主要以APIsample为例，进行扩展说明。

6.1 初始化 DLL

步骤 1	加载 WLAN.Tester.API.dll
步骤 2	调用 WT_DLLInitialize()
备注	■ 使用 WLAN.Tester.API.dll 中其他任何接口之前，调用 WT_DLLInitialize()；而通常，WT API 程序也被封装在固定对象中，可以在对象的初始化时(甚至对象初始化之前)即执行该接口 ■ 如果未调用初始化，则再后续处理中，很可能因为资源未初始化而返回错误码：WT_ERR_CODE_UNKNOW_PARAMETER。

代码示例：

```
int main(int argc , char *argv[])
{
    //初始化DLL环境
    TesterController::Initialize();
    TesterController *tester = new TesterController();
```

```
    //... 其他处理
}
```

6.2 连接仪器

根据使用场景介绍：乒乓测试，独占连接测试（包含多连接版本）

6.2.1 乒乓测试场景：

A电脑IP	192.168.10.1
B电脑IP	192.168.10.2
无线网络测试仪IP	192.168.10.254
场景1	A电脑已经连接WT测试仪 ,此时B电脑调用WT_Connect尝试连接仪器，API会提示仪器已经给A电脑占用。此时B电脑需要轮询查看WT仪器是否被占用
场景2	如果A和B都没有连接WT测试仪，则A和B同时竞争连接。
场景3	如果A没有连接仪器，则B可以直接连接仪器。

实例伪代码如下：

```
int try_connect_tester()
{
    const char *test_ip = "192.168.10.254";
    const char *busy_status = "BUSY";
```

```
const char *idle_status = "IDLE";
const char *err_status = "ERR";
int ret = -1;
char buffer[256] = {0};
int connect_id = -1;
while (1)
{
    ret = WT_GetTesterConnStataus(test_ip,buffer,sizeof(buffer));
    if (!ret && buffer[0])
    {
        if (!strncmp(buffer,idle_status,strlen(idle_status)))//idle
        {
            ret = WT_Connect(test_ip,&connect_id);
            if (WT_ERR_CODE_OK == ret)
            {
                break;//connect ok
            }
        }
        if (!strncmp(buffer,err_status,strlen(err_status)))//error
        {
            break;//connect error
        }
```

```
            }
            Sleep(10);
        }
        return ret;
}
```

6.2.2 强制连接场景

A电脑IP	192.168.10.1
B电脑IP	192.168.10.2
无线网络测试仪IP	192.168.10.254
强制连接	如果A已经连接WT仪器，此时B调用WT_ForceConnect连接仪器，此时仪器会把A的连接断开，B电脑成功连接WT测试仪

6.3 频偏校准

步骤1	配置DUT信号参数
步骤2	根据“步骤1”的参数，配置VSA参数。
步骤3	■ DUT 发送数据帧 ■ 仪器采集数据并分析 WT_RES_FREQ_ERR 对应的结果

	■ 进行帧频偏校准 ■ 校准成功，跳转到步骤 6 ■ 校准不在合理范围，跳转到步骤 5
步骤4	■ 配置 DUT 发送载波信号 ■ 仪器采集数据，取得 WT_RES_SPECTRUM_PEAK_FREQ 对应的结果 ■ 进行载波频偏校准 ■ 校准成功(50ppm 以内)，跳转到步骤 3
步骤5	调整DUT参数，重复步骤3~步骤4.如果超过重试上限，返回失败
步骤6	校准结束
备注	配置VSA参数时，需要根据DUT的配置参数，适当的配置仪器的参考电平和Trigger电平。其中参考电平应该大于等于DUT发送的最大功率，Trigger电平需要设置成最小触发功率

伪代码如下

```
int FreqFrameCalibration()
{
    int ret = 0;

    set_dut_start_tx_frame();
    WT_DataCapture(connect_id);
    //get current frame freq ppm
    ret = WT_GetResult(connect_id,WT_RES_SPECTRUM_PEAK_POW_FREQ);
```

```
    if (pass)
    {
        return 1;
    }
    return 0;
}

int FreqCarrierCalibration()
{
    int ret = 0;
    double dCurrentErr_ppm = 0;
    set_dut_start_tx_carrier();
    WT_DataCapture(connect_id);
    //get current carrier freq ppm
    ret = WT_GetResult(connect_id,WT_RES_SPECTRUM_PEAK_POW_FREQ);
    if (pass)
    {
        return 1;
    }
    return 0;
}
```

```
int FreqCalibration()
{
    int ret = -1;
    int is_carrier = 0;
    int max_power_level = 18;
    int trriger_power_level = -34;
    int is_trriger = 0;
    do
    {
        ret = set_dut_signal();//set dut signal parm
        ret = set_dut_tx_power();//set dut tx power level
        if (ret)
        {
            //set tester parm
            ret = set_tester_rx(is_trriger,max_power_level,trriger_power_level);

            if (!FreqFrameCalibration())//we do frame calibration first
            {
                if (!is_carrier)
                {
                    ret = FreqCarrierCalibration();//we do carrier calibration
```

```
                    }
                    continue;//contniue freq calibration
                }
                break;//freq calibration ok
            }
        } while (0);

        return ret;
}
```

6.4 功率校准

步骤1	配置DUT信号参数
步骤2	根据“步骤1”的参数，配置VSA参数
步骤3	DUT发送数据帧
步骤4	仪器采集数据成功则并分析WT_RES_RMS_DB_NO_GAP对应的结果
步骤5	■ 判断是否校准成功(正负 1dB)。 ■ 成功跳转到步骤 6 ■ 失败则改变 DUT 参数，跳转到步骤 3
步骤6	结束校准

备注	参考电平配置和Trigger电平配置同频偏校准

实现伪代码

```
int get_frame_power()
{
    int ret = 0;
    //capture dut singnal
    WT_DataCapture(connect_id);
    //get frame count
    Ret = WT_GetResult(WT_RES_POWER_FRAME_CNT);
    //get current power
    ret = WT_GetResult(WT_RES_POWER_FRAME_DB);
    if (pass)//check if pass or fail
    {
        return 1;
    }
    return 0;
}


int PowerCalibration()
{
    int ret = -1;
    int is_carrier = 0;
```

```
int max_power_level = 18;
int trriger_power_level = -34;
int is_trriger = 0;
do
{
    ret = set_dut_signal();//set dut signal parm
    ret = set_dut_tx_power();//set dut tx power level
    if (ret)
    {
        ret = set_tester_rx(
            is_trriger,
            max_power_level,
            trriger_power_level
            );//set tester parm

        if (!get_frame_power())//get power
        {
            continue;//continue   calibration
        }
        break;// calibration ok
    }
} while (0);
```

```
    return ret;
}
```

6.5 TX Verify

步骤1	配置DUT信号参数
步骤2	根据“步骤1”的参数，配置VSA参数
步骤3	DUT发送数据帧
步骤4	仪器采集数据成功则并分析各种参数，跳转到步骤6
步骤5	采集数据失败或者分析失败，跳转到步骤3
步骤6	判断是否成功(正负2dB)。注：这里的2dB可以自定义判断标准
备注	配置DUT信号参数时，可以从本地配置文件里面读取设置DUT Tx Packet数量，信号的gap值，retry次数，Tx Verify需要判断哪些信息等。一般情况采集完信号后，分析的参数有： **WT_RES_POWER_FRAME_DB** **WT_RES_FRAME_EVM_ALL** **WT_RES_FRAME_FREQ_ERR** **WT_RES_SPECTRUM_MASK_ERR_PERCENT** 具体分析哪几个选项，可以自己定制
伪代码实现	

```
int get_dut_tx_all()
{
    int ret = 0;
    WT_DataCapture();
    //get power
    ret = WT_GetResult(WT_RES_POWER_FRAME_DB);
    //get freq error ppm
    ret = WT_GetResult(WT_RES_FRAME_FREQ_ERR);
    //get EVM
    ret = WT_GetResult(WT_RES_FRAME_EVM_ALL);
    // get spectrum mask error
    ret = WT_GetResult(WT_RES_SPECTRUM_MASK_ERR_PERCENT);
    if (pass)//check if pass or fail
    {
        return 1;
    }
    return 0;
}



int tx_verify()
{
```

```
int ret = -1;
int is_carrier = 0;
int max_power_level = 18;
int trriger_power_level = -34;
int is_trriger = 0;
do
{
    ret = set_dut_signal();//set dut signal parm
    ret = set_dut_tx_power();//set dut tx power level
    if (ret)
    {
        //set tester parm
        ret = set_tester_rx(is_trriger,max_power_level,trriger_power_level);

        if (!get_dut_tx_all())//get power
        {
            continue;//continue
        }
        break;//ok
    }
} while (0);
```

```
    return ret;
}
```

6.6 RX Verify

步骤1	配置DUT参数，进入PER测试状态
步骤2	根据“步骤1”的参数，配置VSG参数。
步骤3	VSG发送数据帧
步骤4	DUT分析PER等参数
步骤5	判断是否PASS标准

伪代码实现

```
const int vsg_tx_packet = 1000;


int get_dut_rx_per()
{
    int dut_rx_packet_count = get_dut_packet();
    double per_pass_present = dut_rx_packet_count*100/vsg_tx_packet;
    double per_fail_present =
(vsg_tx_packet-dut_rx_packet_count)*100/vsg_tx_packet;
    if (pass)
    {
```

```
            return 1;
        }
        return 0;
}


int rx_verify()
{
        int ret = -1;
        int is_carrier = 0;
        int max_power_level = 18;
        int trriger_power_level = -34;
        int frame_count = 0;
        do
        {
            ret = set_dut_signal();//set dut signal parm
            ret = set_dut_rx_start();//set dut tx power level
            if (ret)
            {
                ret = WT_SetVSG();
                ret = WT_StartVSG();
                if (!get_dut_rx_per())//get power
                {
```

```
                    continue;//continue
                }
                break;//ok
            }
        } while (0);

        return ret;
}
```

6.7 断开仪器

乒乓测试场景	为了提高测试效率使其他等待连接仪器的电脑能连接上 WT 测试仪，如果后续的 DUT 测试步骤已经不需要用到 WT 测试仪，此时调用 **WT_DisConnect** 可以断开仪器。
其他场景	调用**WT_DisConnect**可以断开仪器

6.8 释放 DLL

释放 DLL	程序不再使用 WLAN.Tester.API.dll 中的接口，则可以在在释放该 dll 句柄前调用 **WT_DLLTerminate**();
备注	通常产测程序为专用的测试程序，即该程序的主要目的即为通过

	WLAN.Tester.API.dll 以及相应的 DUT 控制指令，完成 DUT 的出厂生产过程，所以一般来说，也可以将 WT API 的释放接口 **WT_DLLTerminate** 在退出 APP 时进行