对外平台开发手册

2018.10.30

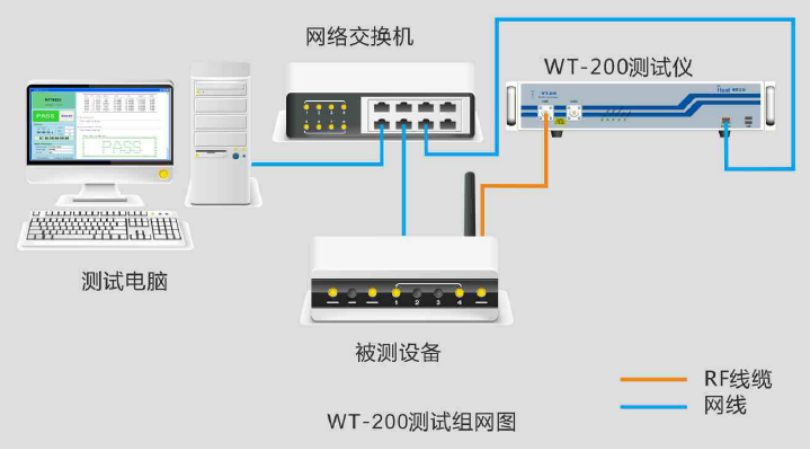
目录

一、如何搭建开发环境

1.1 对外平台开发条件

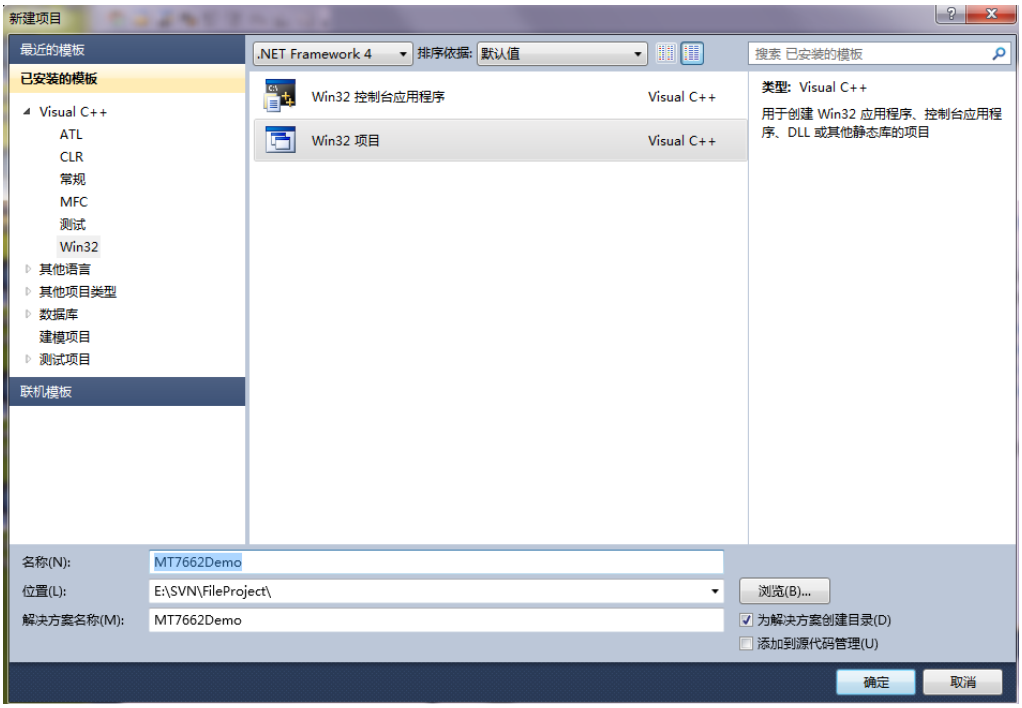
1. WT 系列仪器一台
2. VS2010 以上版本的软件
3. 被测设备一块（本文以 MT7662 为例子讲述）
4. 开发要求的库文件，统一存放于 Platform 文件夹
5. 开发要求的头文件，包括 Signal.h、DutBase.h、CallBackFunc.h、IDut.h、DutDefine.h、DutDefine.cpp（只需要修改 DutDefine.h，其余文件不可修改）

1.2 组网图

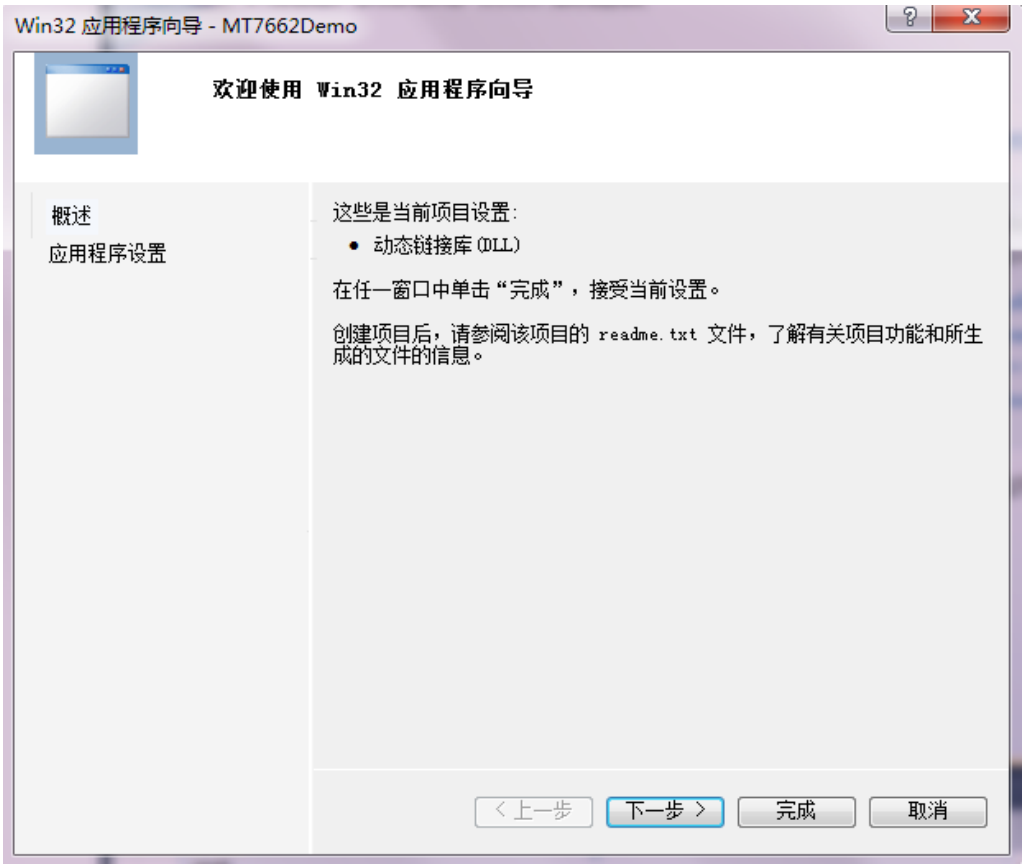


1.3 工程建立

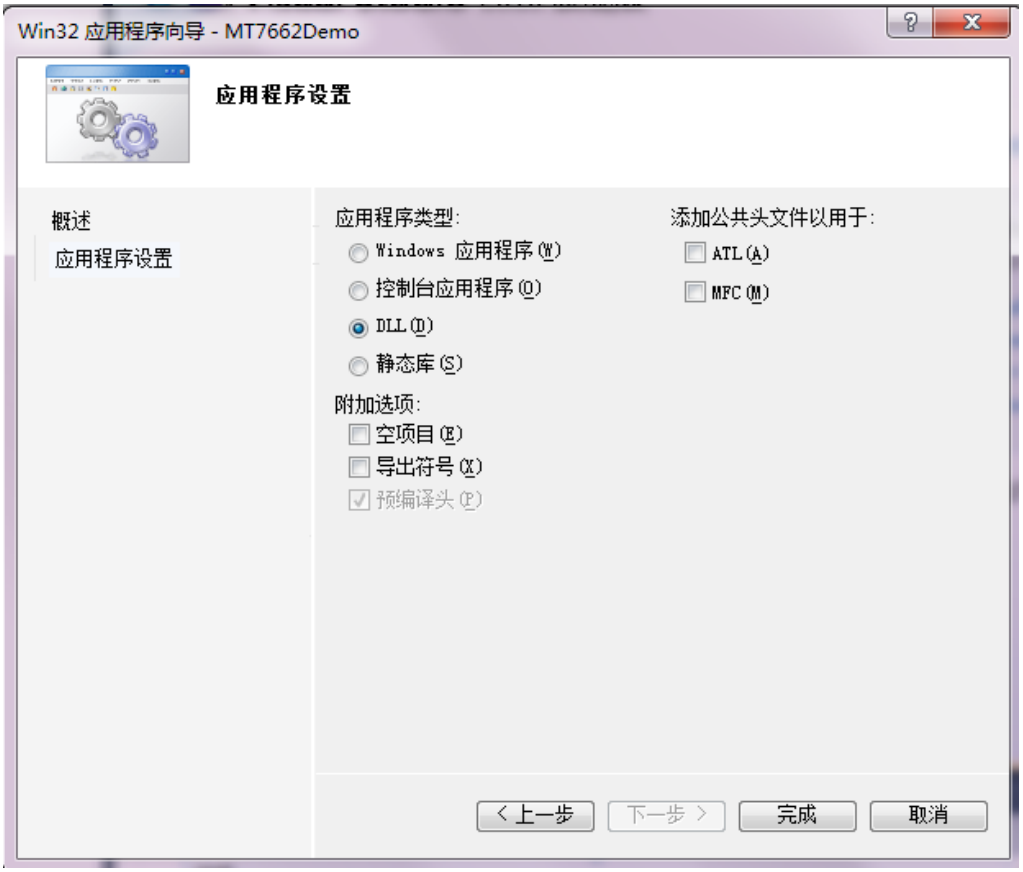
1）在 VS2010 建立了一个 MT7662Demo 的 dll 工程



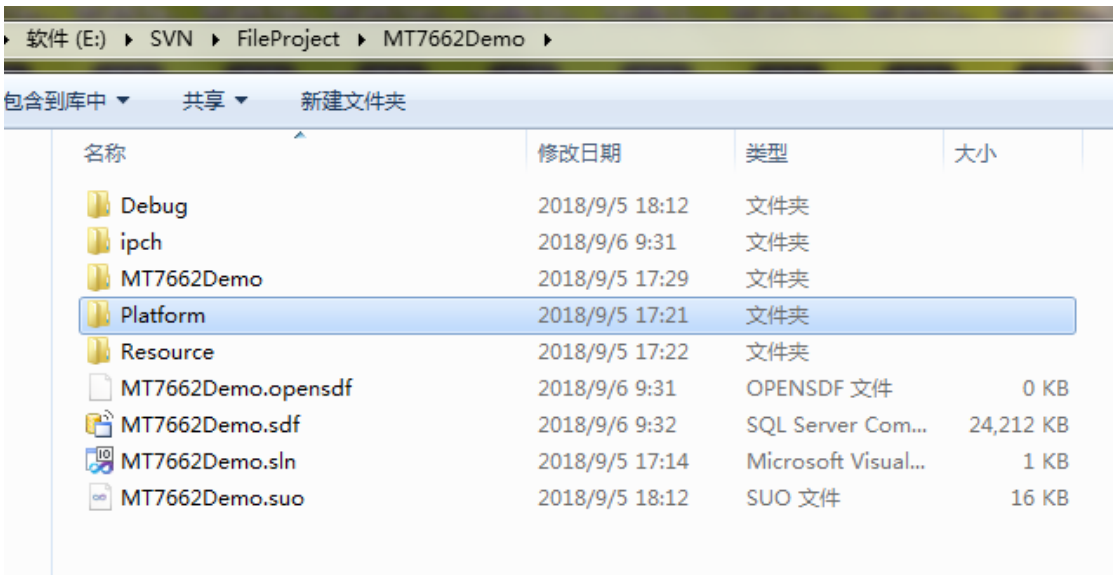
设定是动态链接库，点击完成，如果不是则按下一步设定



选择应用程序类型为 DLL，点击完成

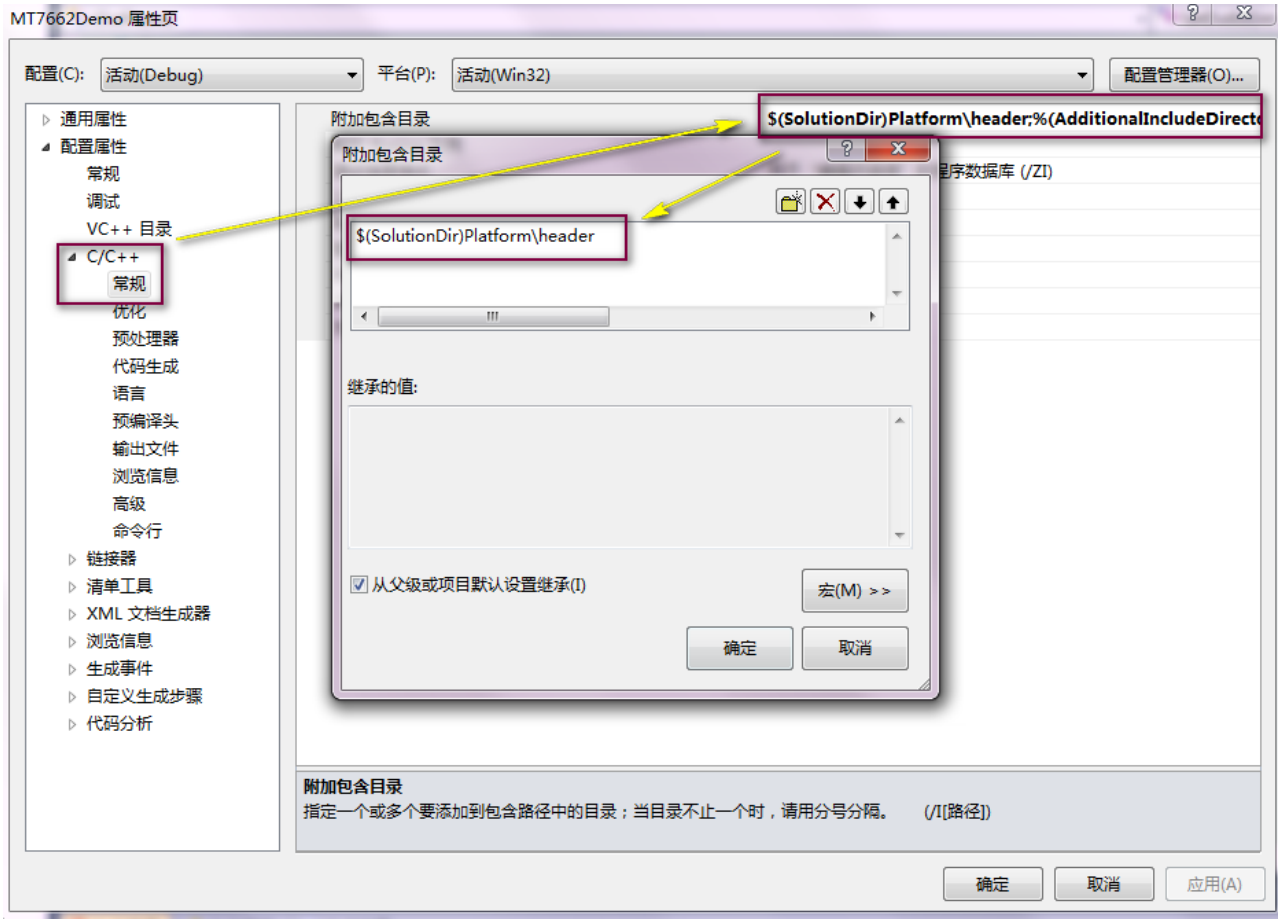


2）把平台的 Platform 文件夹放入工程目录中，Platform 中含有平台的库文件以及要用到的头文件

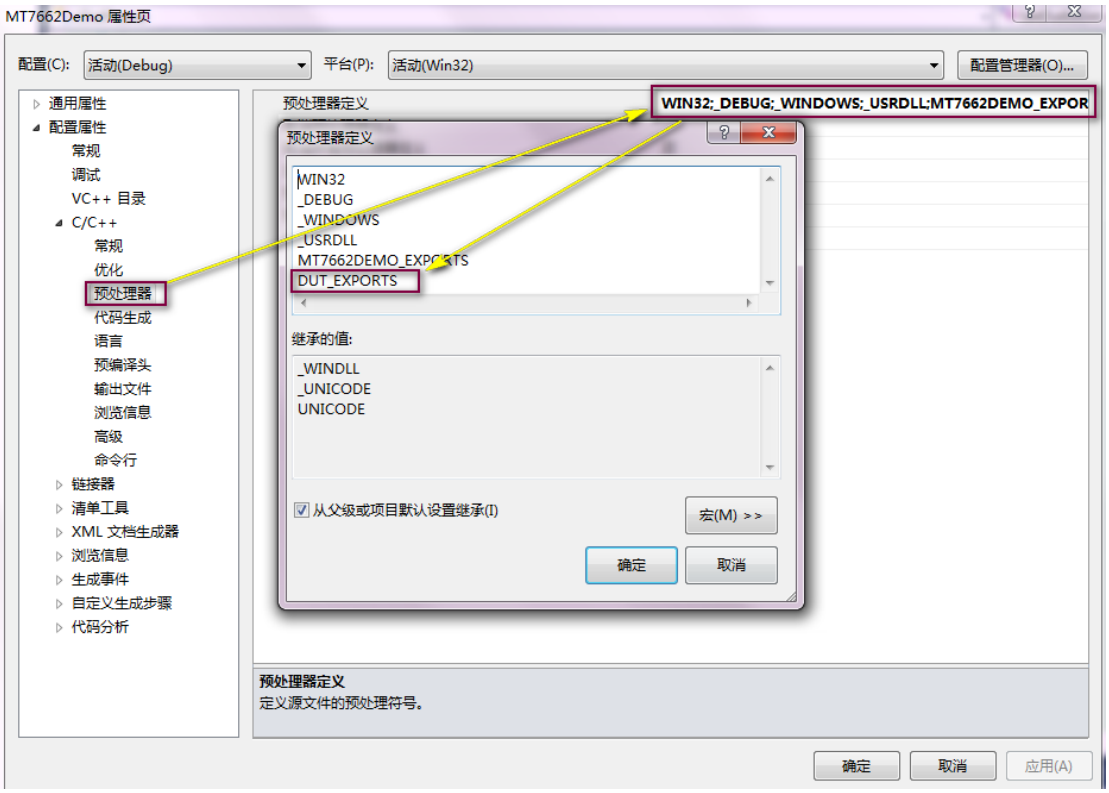


3）把 Platform\DLL\Debug 路径下的所有文件夹以及文件复制到 Debug 里，把 Platform\resource 中的两个文件复制到 MT7662Demo 文件夹中

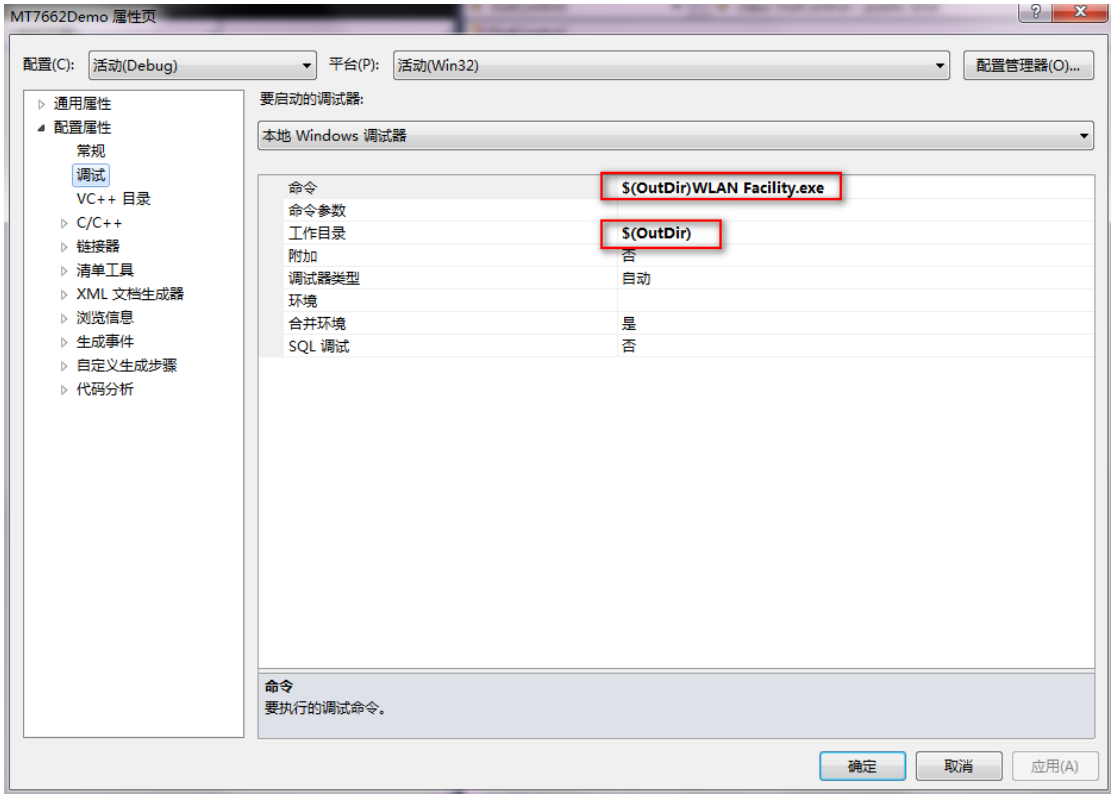
4）设定包含目录，关联平台提供的头文件



5）添加预处理定义



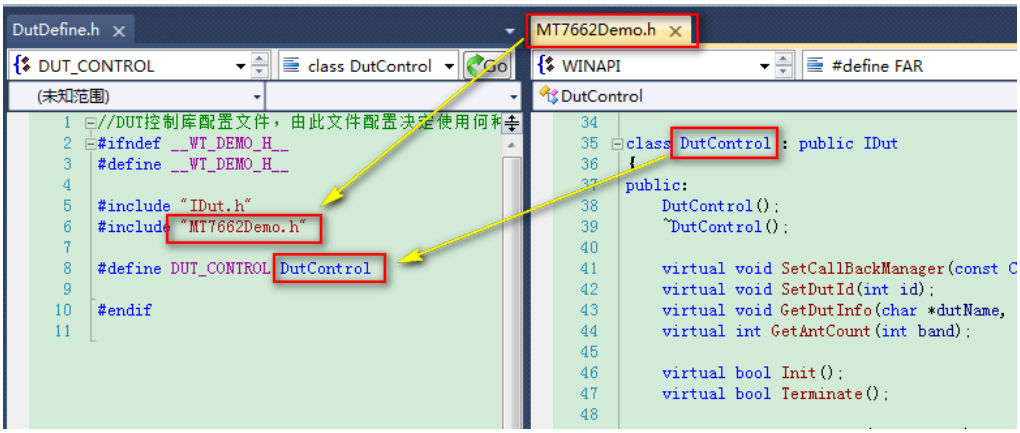
6）修改调试命令以及工作目录



7）使用IDut.h定义的命名空间DUT_NAMESPACE，创建新类，继承IDut接口类并实例化所有函数，若实现过程有不需要使用的函数直接返回或返回true

1.4 实现插件与平台对接

1）只需要修改提供的DutDefine.h文件中的头文件以及宏定义，以MT7662为例子



把继承接口类的头文件名称以及继承接口类的类名修改至DutDefine.h的指定位置

2）开发完毕后，把生成的DUT插件dll的名称填入PluginCfg.xml中相应的位置，不然平台无法识别要加载的DUT插件，如下图：

二、快速建立模型

根据第一章的方法搭建好开发环境后，快速编译一个最小化测试程序，了解产测、DUT、仪器的交互流程。只需要实现五个接口，Open、Close、TxFrame、TxStop、GetAntCount。其中 GetAntCount 仅需要根据实际返回板子拥有的天线数

1 第一步实现与 DUT 建立通讯

在 virtual bool Open(int type)接口实现打开 DUT，并进行 DUT 的初始化操作，正式进入可以发包的状态。7668 例子如下：

```
//连接DUT，执行指定的初始化命令
bool DutControl::Open(int type)
{
    bool ret = false;

    //加载dll的函数
    m_libHandle = LoadLibraryA("HQADLL.dll");

    m_pbOpenAdapter = (bool_Func_void)GetProcAddress(m_libHandle, "?HQA_OpenAdapter@@YGHXZ");
    m_pbCloseAdapter = (bool_Func_void)GetProcAddress(m_libHandle, "?HQA_CloseAdapter@@YGHXZ");
    m_pbStartTx = (bool_Func_ushort)GetProcAddress(m_libHandle, "?HQA_StartTx@@YGHKG@Z");
    m_pbTxStop = (bool_Func_void)GetProcAddress(m_libHandle, "?HQA_StopTx@@YGHXZ");
    m_pbSetChannel = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetChannel@@YGHE@Z");
    m_pbSetDemode = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetPreamble@@YGHE@Z");
    m_pbSetRate = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetRate@@YGHE@Z");
    m_pbSetFreqreg = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetFreqOffset@@YGHE@Z");
    m_pbSetNss = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetNss@@YGHE@Z");
    m_pbPerPktBW = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetPerPktBW@@YGHE@Z");
    m_pbSetPrimaryBW = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetPrimaryBW@@YGHE@Z");
    m_pbSetSystemBW = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetSystemBW@@YGHE@Z");
    m_pbSetTxIPG = (bool_Func_u_long)GetProcAddress(m_libHandle, "?HQA_SetTxIPG@@YGHK@Z");
    m_pbSetTxPath = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetTxPath@@YGHE@Z");
    m_pbSetTssiOnOff = (bool_Func_uchar)GetProcAddress(m_libHandle, "?HQA_SetTssiOnOff@@YGHE@Z");//0:off  1:on
    m_pbSetATEMode = (bool_Func_bool)GetProcAddress(m_libHandle, "?HQA_SetATEMode@@YGHH@Z");

    do
    {
        if (!m_pbOpenAdapter())
        {
            break;
        }

        if (!m_pbSetATEMode(true))
        {
            break;
        }

        if (!m_pbSetTssiOnOff(1))
        {
            break;
        }

        ret = true;
    } while (0);

    return ret;
}
```

2 第二步实现 DUT 发送信号

在 virtual bool TxFrame(int txmode = TX_MODE_COMMON)接口实现 DUT 的发包操作，固定 DUT 发送频道为 2412，11g 54M，由天线 0 发出的信号，设定无限发包。7668 例子如下：

```
//控制DUT发送2412 11G 54M 天线0的信号
bool DutControl::TxFrame(int txmode /*= TX_MODE_COMMON*/)
{

    bool bRslt = false;
    int m_demodeValue[eumDemodMax];

    m_demodeValue[eumDemod_11b] = 0;
    m_demodeValue[eumDemod_11ag] = 1;
    m_demodeValue[eumDemod_11n_20M] = 2;
    m_demodeValue[eumDemod_11n_40M] = 2;
    m_demodeValue[eumDemod_11ac_20M] = 4;
    m_demodeValue[eumDemod_11ac_40M] = 4;
    m_demodeValue[eumDemod_11ac_80M] = 4;
    DutSignal m_signal;
    m_signal.channel = 2412;
    m_signal.bw = 20;
    m_signal.demode = eumDemod_11ag;
    m_signal.mcs = 6;

    do
    {
        if (!m_pbSetChannel(m_signal.channel))
        {
            break;
        }

        if (!m_pbSetSystemBW(m_signal.bw))
        {
            break;
        }

        if (!m_pbPerPktBW(m_signal.bw))
        {
```

3 第三步实现 DUT 停止发送信号

在 virtual bool TxStop()接口实现控制 DUT 停止发送信号。7668 例子如下：

```
//控制DUT停止发包
bool DutControl::TxStop()
{
    return m_pbTxStop();
}
```

4 第四步实现 DUT 的断开连接

在 virtual bool Close(int type)接口实现 DUT 连接的断开，销毁所有的资源.。7668 例子如下：

```
//断开DUT，释放资源
bool DutControl::Close(int type)
{
    bool bRlst = true;

    bRlst &= m_pbSetATEMode(false);
    bRlst &= m_pbCloseAdapter();

    FreeLibrary(m_libHandle);
    m_libHandle = NULL;

    return bRlst;
}
```

此时已经完成最小化产测，可以调试运行完整的产测（参考 MT7662Simple 文件夹工程）。

三、软件流程关系

3.1 软件如何运行

WLAN Facility.exe 的运行顺序的依据是 FLOW 的顺序，举个例子，如下图：

```
//*****************************************************
WT_CONNECT_TESTER                //连接测试仪

WT_INSERT_DUT                    //连接DUT

WT_VERIFY_TX_ALL     1    54M     CHAIN1
WT_VERIFY_RX_PER     1    54M     CHAIN1

WT_REMOVE_DUT                    //关闭DUT

WT_DISCONNECT_TESTER             //断开仪器连接
//*****************************************************
```

软件的运行的流程根据上图就是连接仪器、连接 DUT、TX 测试、RX 测试、断开 DUT、断开仪器。每一行的 FLOW 都是一个单独的操作，但是 FLOW 之间有互相的依赖关系。比如 TX 测试必须先连接仪器和连接 DUT 才可以执行。仪器部分均不需要实现，用户只需要实现 DUT 的控制部分。

所以软件最终的执行方案是，检测 FLOW 的项目，形成一个运行的顺序表，然后按顺序表运行 FLOW 对应的模块。每个模块对应着调用不同的接口，模块与接口的关系是固定的。

3.2 流程-flow-接口对应表

流程	接口	FLOW
打开 DUT	Open	WT_INSERT_DUT
关闭 DUT	Close	WT_REMOVE_DUT
校准开始	CalBegin	WT_CAL_START
校准结束	CalEnd	WT_CAL_END
帧信号频偏校准	GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxFrame -> AdjustFreqReg -> TxStop	WT_CAL_FREQ_FRAME
单载波信号校准	GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxCarrier -> AdjustFreqReg_CW -> TxCarrierStop	WT_CAL_FREQ_CW
功率校准	GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxFrame -> AdjustPwrReg -> TxStop	WT_CAL_PWR
TX 测试	GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxFrame -> TxStop	WT_VERIFY_TX_ALL
RX 测试	GetAntCount -> SetSignal -> RxStart -> GetPerResult -> RxStop	WT_VERIFY_RX_PER
读取温补值	TempCompensation	WT_READ_THERMAL
写 MAC 地址	SetMac -> GetMac	WT_SET_MAC_ADDRESS

写入校准数据	SaveCalData	WT_SAVE_CAL_DATA
写入校准数据	WriteEfuse	WT_WRITE_EFUSE_FREE
检查校准数据	CheckEfuse	WT_CHECK_EFUSE_WRITE
判断 DUT 是否 Ready	IsDutReady	
获取界面显示名字及版本号	GetDutInfo	
设置平台资源	SetDutId -> SetCallBackManager	

（根据实际需求决定运行哪一项 FLOW，对应着实现 FLOW 相关接口，剩余的 FLOW 对应的接口直接返回即可，不影响其他的 FLOW。没有对应 FLOW 的流程为平台固定调用的流程，根据需要实现，不实现也不影响）

四、如何建立与 DUT 的通讯

4.1 初始化与 DUT 建立通讯

FLOW 配置示例：

```
//*****************************************************
WT_CONNECT_TESTER                   //连接测试仪

WT_INSERT_DUT                       //连接DUT

WT_VERIFY_TX_ALL     1    54M       CHAIN1
WT_VERIFY_RX_PER     1    54M       CHAIN1

WT_REMOVE_DUT                       //关闭DUT

WT_DISCONNECT_TESTER                //断开仪器连接
//*****************************************************
```

建立DUT通讯与FLOW配置中的WT_INSERT_DUT和WT_REMOVE_DUT流程关联

平台在运行FLOW和接口调用的关联性如下：

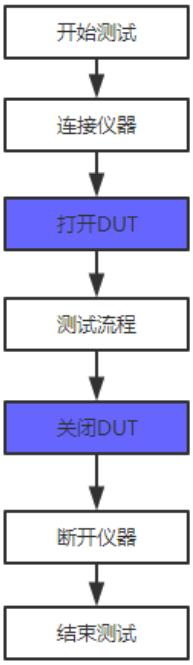
FLOW项	接口函数
WT_INSERT_DUT	virtual bool Open(int type)
WT_REMOVE_DUT	virtual bool Close(int type)

接口参数传入的值由FLOW项的参数决定，例如：

FLOW项	枚举
WT_INSERT_DUT	WT_DUT_GENERICITY
WT_INSERT_DUT 2G	WT_DUT_WIFI_2G
WT_INSERT_DUT 5G	WT_DUT_WIFI_5G
WT_INSERT_DUT BT	WT_DUT_BT
WT_INSERT_DUT ZIGBEE	WT_DUT_ZIGBEE
WT_INSERT_DUT CW	WT_DUT_CW
WT_INSERT_DUT LORA	WT_DUT_LoRa
WT_INSERT_DUT LRWPAN	WT_DUT_Lrwpan

以此类推，WT_REMOVE_DUT同上，提供的值参考代码枚举enum_InsertType

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
打开DUT	Open
关闭DUT	Close

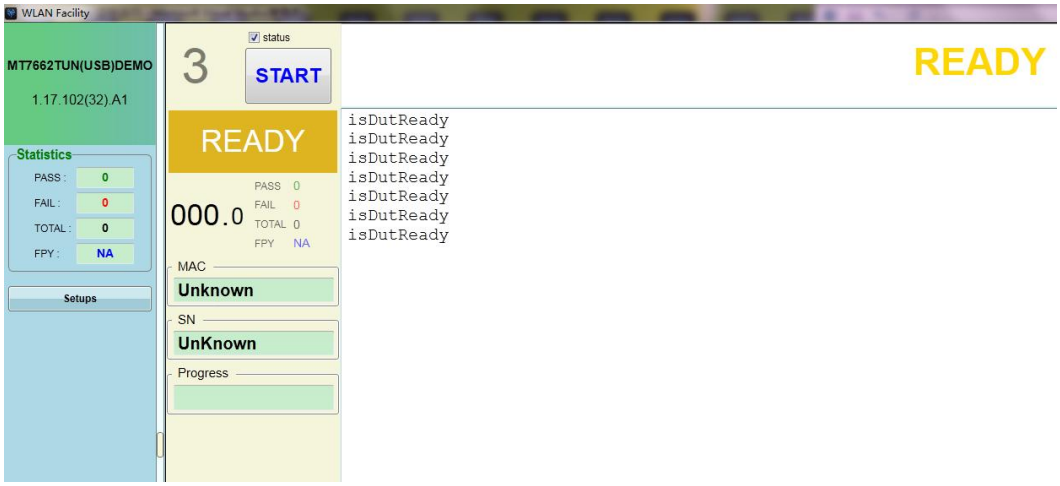
接口被调用流程：

Open -> Close

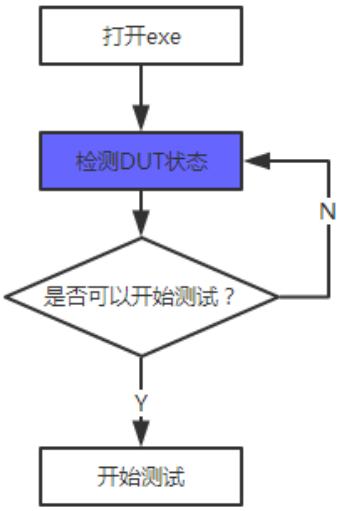
4.2 测试预备标志

打开软件，界面中黑色框位置是显示板子的状态，处于灰色IDLE状态

经过平台调用接口返回true后状态改变为黄色的ready状态



平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
检测DUT状态	IsDutReady

接口被调用流程：

IsDutReady

五、如何实现 TX，RX 测试

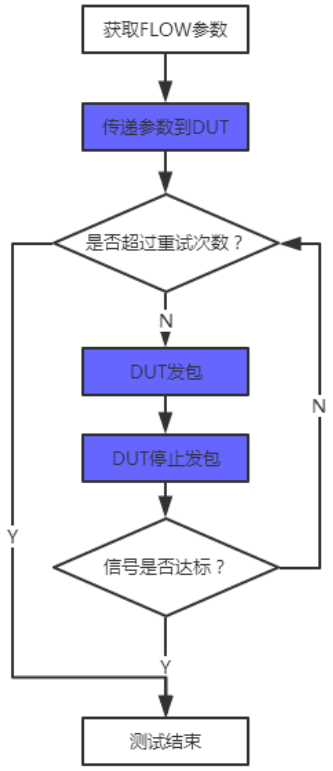
5.1TX 测试

FLOW 配置示例：

```
WT_CONNECT_TESTER                    //连接测试仪
WT_INSERT_DUT                        //连接DUT
WT_VERIFY_TX_ALL     1    54M            CHAIN1
WT_REMOVE_DUT                        //关闭DUT
WT_DISCONNECT_TESTER                 //断开仪器连接
```

WT_VERIFY_TX_ALL 为 TX 流程关键字，所有操作在此关键字完成

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
传递参数到DUT	GetAntCount、SetSignal、GetTgtPwrAndPwrReg
DUT发包	TxFrame
DUT停止发包	TxStop

接口被调用流程：

GetAntCount ->SetSignal->GetTgtPwrAndPwrReg ->TxFrame->TxStop

5.2RX 测试

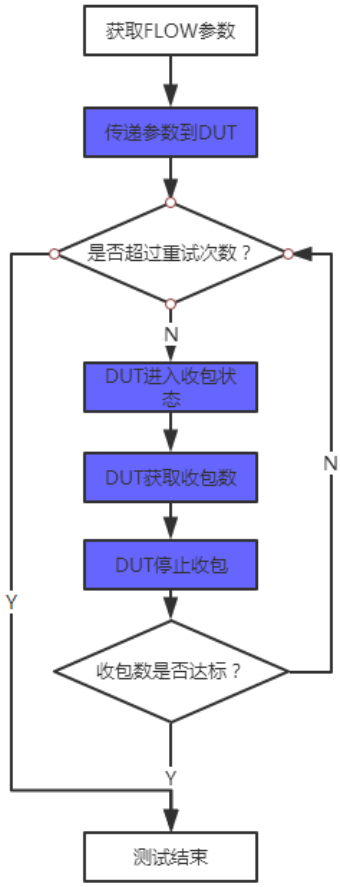
RX 流程(Rx 需要在 wave 文件夹存放波形文件，WT160 使用 80M，WT2XX 使用 120M，WT3XX 使用 240M)

FLOW 配置示例：

```
//
WT_CONNECT_TESTER                  //连接测试仪

WT_INSERT_DUT                      //连接DUT

WT_VERIFY_RX_PER     1    54M      CHAIN1

WT_REMOVE_DUT                          //关闭DUT

WT_DISCONNECT_TESTER                   //断开仪器连接
```

WT_VERIFY_RX_PER 为 RX 流程关键字，所有操作在此关键字完成

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
传递参数到DUT	GetAntCount、SetSignal
DUT开始收包	RxStart
DUT获取收包数	GetPerResult
DUT停止收包	RxStop

接口被调用流程：

GetAntCount ->SetSignal ->RxStart->GetPerResult ->RxStop

六、如何实现校准

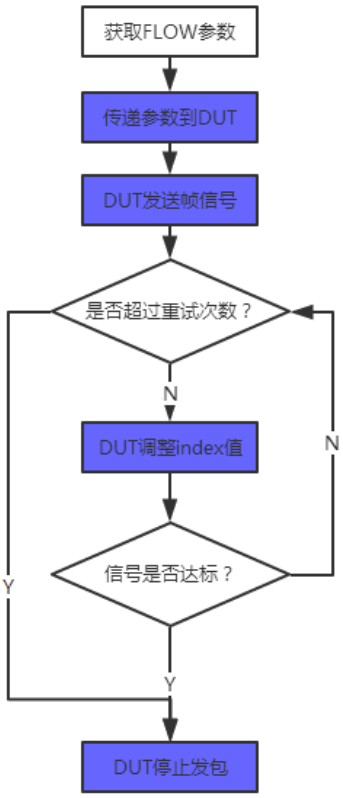
6.1 频偏帧校准

FLOW 配置示例：

```
WT_CONNECT_TESTER                //连接测试仪
WT_INSERT_DUT                    //连接DUT
WT_CAL_START
    WT_CAL_FREQ_FRAME       6    54M CHAIN1
WT_CAL_END
WT_REMOVE_DUT                        //关闭DUT
WT_DISCONNECT_TESTER                 //断开仪器连接
```

WT_CAL_FREQ_FRAME 为帧校准流程的关键字，校准操作都在此关键字完成
WT_CAL_START、WT_CAL_END 为标志项，可以不实现操作

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数

传递参数到DUT	GetAntCount、SetSignal、GetTgtPwrAndPwrReg
DUT发送帧信号	TxFrame
DUT调整index值	AdjustFreqReg
DUT停止发包	TxStop

接口被调用流程：

GetAntCount ->SetSignal -> GetTgtPwrAndPwrReg -> TxFrame -> AdjustFreqReg ->TxStop

6.2 频偏单载波校准

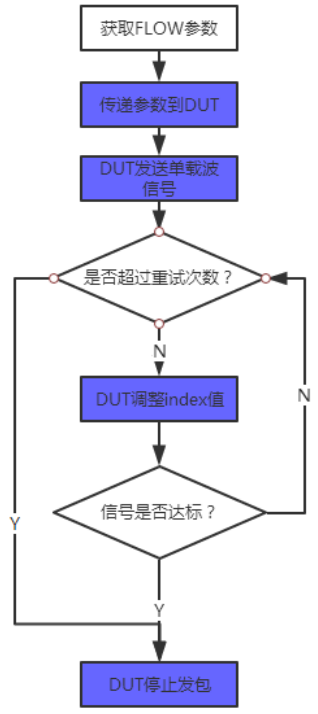
FLOW 配置示例：

```
WT_CONNECT_TESTER                    //连接测试仪
WT_INSERT_DUT                        //连接DUT
WT_CAL_START
    WT_CAL_FREQ_CW      6    54M CHAIN1
WT_CAL_END
WT_REMOVE_DUT                        //关闭DUT
WT_DISCONNECT_TESTER                 //断开仪器连接
```

WT_CAL_FREQ_CW 为帧校准流程的关键字，校准操作都在此关键字完成

WT_CAL_START、WT_CAL_END 为标志项，可以不实现操作

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
传递参数到DUT	GetAntCount、SetSignal、GetTgtPwrAndPwrReg
DUT发送单载波信号	TxCarrier
DUT调整index值	AdjustFreqReg_CW
DUT停止发包	TxCarrierStop

接口被调用流程：

GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxCarrier -> AdjustFreqReg_CW -> TxCarrierStop

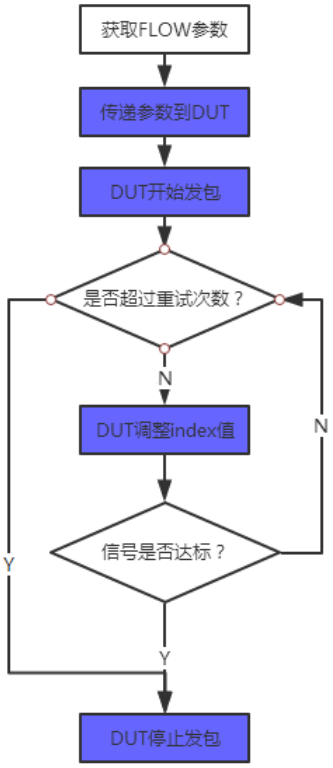
6.3 功率校准

FLOW 配置示例：

```
WT_CONNECT_TESTER               //连接测试仪

WT_INSERT_DUT                   //连接DUT

WT_CAL_START

    WT_CAL_PWR          1   54M CHAIN1
    WT_CAL_PWR          6   54M CHAIN1
    WT_CAL_PWR          11  54M CHAIN1

    WT_CAL_PWR          36  54M CHAIN1
    WT_CAL_PWR          44  54M CHAIN1
    WT_CAL_PWR          52  54M CHAIN1
    WT_CAL_PWR          64  54M CHAIN1

    WT_CAL_PWR          100 54M CHAIN1
    WT_CAL_PWR          106 54M CHAIN1
    WT_CAL_PWR          120 54M CHAIN1
    WT_CAL_PWR          140 54M CHAIN1

    WT_CAL_PWR          149 54M CHAIN1
    WT_CAL_PWR          165 54M CHAIN1

WT_CAL_END

WT_REMOVE_DUT                       //关闭DUT

WT_DISCONNECT_TESTER                //断开仪器连接
```

WT_CAL_PWR 为功率校准流程的关键字，校准操作都在此关键字完成

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
传递参数到DUT	GetAntCount、SetSignal、GetTgtPwrAndPwrReg
DUT开始发包	TxFrame
DUT调整index值	AdjustPwrReg
DUT停止发包	TxStop

接口被调用流程：

GetAntCount -> SetSignal -> GetTgtPwrAndPwrReg -> TxFrame ->
AdjustPwrReg -> TxStop

七、如何实现数据写入

7.1 写入 flash

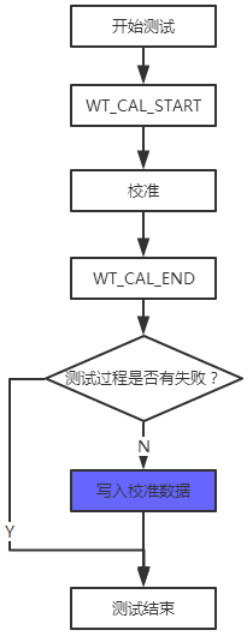
FLOW 配置示例

```
WT_CONNECT_TESTER                //连接测试仪

WT_INSERT_DUT                    //连接DUT

WT_CAL_START

    WT_CAL_FREQ_FRAME    6    54M CHAIN1

    WT_CAL_PWR           1    54M CHAIN1
    WT_CAL_PWR           6    54M CHAIN1
    WT_CAL_PWR           11   54M CHAIN1

    WT_CAL_PWR           36   54M CHAIN1
    WT_CAL_PWR           44   54M CHAIN1
    WT_CAL_PWR           52   54M CHAIN1
    WT_CAL_PWR           64   54M CHAIN1

    WT_CAL_PWR           100  54M CHAIN1
    WT_CAL_PWR           106  54M CHAIN1
    WT_CAL_PWR           120  54M CHAIN1
    WT_CAL_PWR           140  54M CHAIN1

    WT_CAL_PWR           149  54M CHAIN1
    WT_CAL_PWR           165  54M CHAIN1

WT_CAL_END

WT_SAVE_CAL_DATA

WT_REMOVE_DUT                        //关闭DUT

WT_DISCONNECT_TESTER                 //断开仪器连接
```

WT_SAVE_CAL_DATA 为写值流程的关键字，写数据到 flash 操作都在此关键字完成

注意写值操作是有前提要求的，平台要求必须执行了 WT_CAL_START 和 WT_CAL_END 并且测试过程不失败才会执行 WT_SAVE_CAL_DATA 操作

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
写入校准数据	SaveCalData

接口被调用流程：

SaveCalData

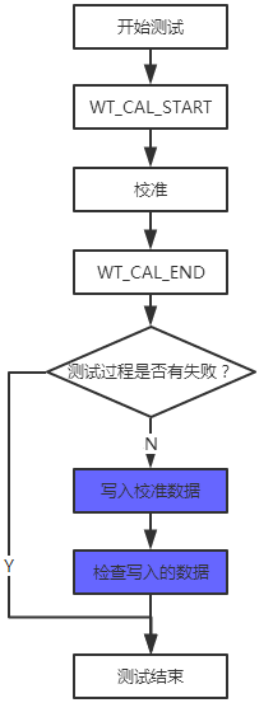
7.2 写入 efuse

FLOW 配置示例

```
WT_CONNECT_TESTER                //连接测试仪

WT_INSERT_DUT                    //连接DUT

WT_CAL_START

    WT_CAL_FREQ_FRAME    6    54M CHAIN1

    WT_CAL_PWR           1    54M CHAIN1
    WT_CAL_PWR           6    54M CHAIN1
    WT_CAL_PWR           11   54M CHAIN1

    WT_CAL_PWR           36   54M CHAIN1
    WT_CAL_PWR           44   54M CHAIN1
    WT_CAL_PWR           52   54M CHAIN1
    WT_CAL_PWR           64   54M CHAIN1

    WT_CAL_PWR           100  54M CHAIN1
    WT_CAL_PWR           106  54M CHAIN1
    WT_CAL_PWR           120  54M CHAIN1
    WT_CAL_PWR           140  54M CHAIN1

    WT_CAL_PWR           149  54M CHAIN1
    WT_CAL_PWR           165  54M CHAIN1

WT_CAL_END

WT_WRITE_EFUSE_FREE

WT_CHECK_EFUSE_WRITE

WT_REMOVE_DUT                        //关闭DUT

WT_DISCONNECT_TESTER                 //断开仪器连接
```

WT_WRITE_EFUSE_FREE 为写值流程的关键字，写数据到 efuse 操作都在此关键字完成注意写值操作是有前提要求的，平台要求必须执行了 WT_CAL_START 和 WT_CAL_END 并且测试过程不失败才会执行 WT_WRITE_EFUSE_FREE 操作。另外与写入 flash 有所区别的地方是，这里提供多一个 FLOW 项 WT_CHECK_EFUSE_WRITE 来检查写入的值是否生效

平台操作流程：



（蓝色框为需要实现的流程）

平台操作流程与接口被调用流程对应关系：

流程	接口函数
写入校准数据	WriteEfuse
检查写入的数据	CheckEfuse

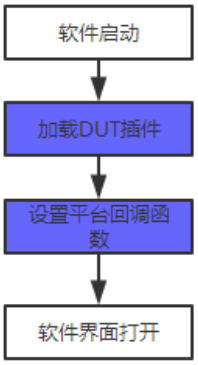
接口被调用流程：

WriteEfuse -> CheckEfuse

附录、如何使用平台提供的方法

使用平台提供的资源

平台操作流程：



（蓝色框为需要实现的流程）

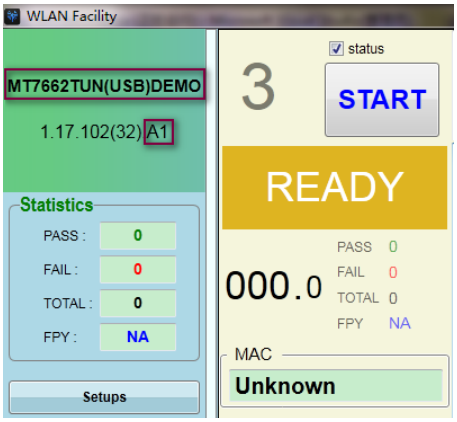
平台操作流程与接口被调用流程对应关系：

流程	接口函数
加载DUT插件	GetDutInfo
设置平台回调函数	SetDutId、SetCallBackManager

接口被调用流程：
GetDutInfo -> SetDutId -> SetCallBackManager

接口说明（详细参考 CallBackFunc.h）：
1）virtual void GetDutInfo(char *dutName, int nameSize, char *version, int versionSize) = 0;

获取显示到界面上的型号与版本号，如下图所示



dutName 为图中 MT7662TUN(USB)DEMO，version 为图中 A1，1.17.102(32)为平台版本不可修改。

2）virtual void SetDutId(int id) = 0;

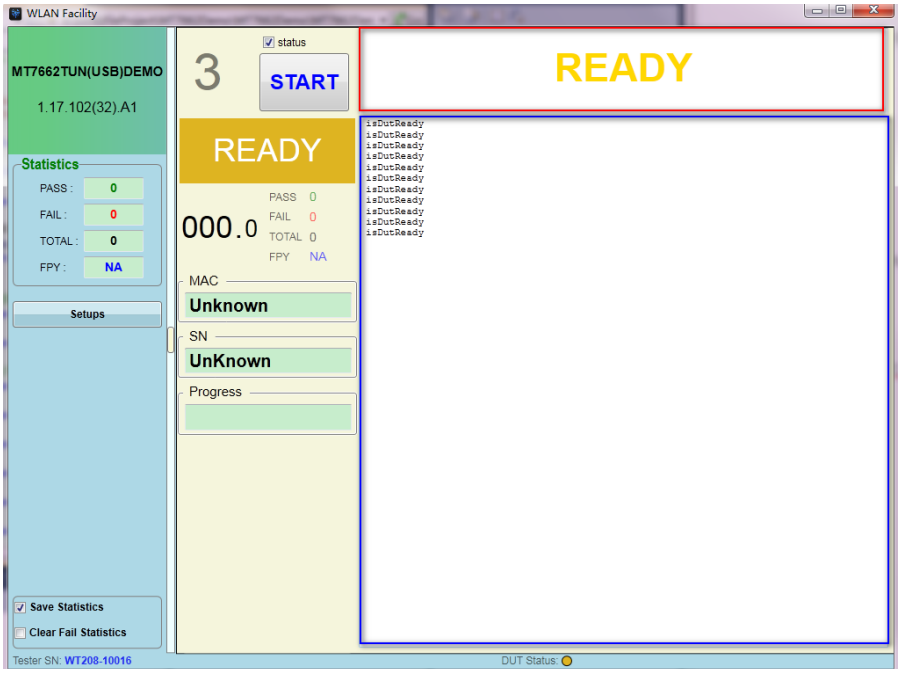
设置当前 DUTid，用以识别当前操作的 DUT 是哪一块 DUT，使用仪器的第几个端口就是下传的 id。目前最多支持一拖八，id 为 1~8

3）typedef void (*PrintLogCB)(int dutId, int printfType, const char *printfContent);

打印 log 到界面的回调函数，如果需要额外打印数据到 log 可以使用此接口。

dutId	SetDutId 接口下传的值
printfType	打印的用途以及输出区域。参考 CallBackFunc.h 中枚举 PrintType
printfContent	要打印的内容

示例：



Enum_Title 打印内容输出到红色框区域

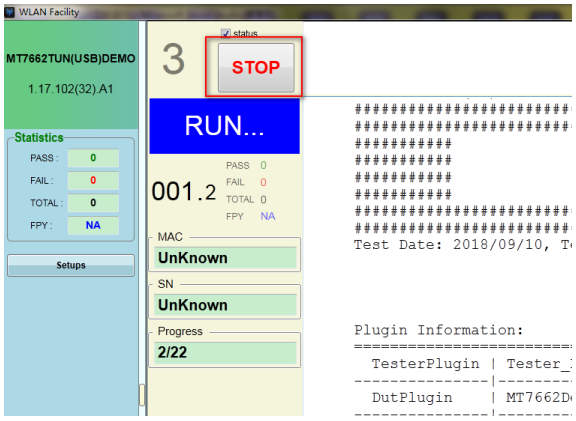
Enum_Text 打印内容以黑色字体输出到蓝色框区域

Enum_Error 打印内容以红色字体输出到蓝色框区域

Enum_Debug 打印内容输出到软件目录 LOG 中的 DEBUG 文件夹，不输出到界面

4） bool (*IsStopRunningCB)(int dutId);

获取 Stop 标志，判断是否点击了 stop 按钮的回调函数



返回 STOP 按钮处的状态，未开始测试：START 返回 false，测试中：STOP 返回 true

5）typedef bool (*CfgElementOperationCB)(int dutId, int opType, int cfgType, const char* key, char* value, int valueSize);

读写配置容器回调函数，只能读取仅有单个值的关键字。在软件开启的时候，平台会读取软件目录下 WT_SETUP 的所有文件内容，可以通过此接口快速读取其中的内容，也可以按文件中的格式自行添加关键字和数值，使用相应的关键字在软件中读取。只能读取写入 WT_BT_LIMIT、WT_DUT_MIMO、WT_MAC、WT_TESTER、WT_WIFI_LIMIT、WT_ZIGBEE_LIMIT

dutId	SetDutId 接口下传的值
opType	读取或是写入。参考 CallBackFunc.h 中枚举 OperateType
cfgType	操作的文件。参考 CallBackFunc.h 中枚举 CfgType
key	文件中的关键字
value	关键字后的跟着的值
valueSize	读取时 char 数组的大小

6)typedef bool (*CfgArrayElementOperationCB)(int dutId, int opType, int cfgType, const char* key, char* value, int valueSize, int elemIndex);

读写数组配置容器回调函数，只能读取拥有多个值的关键字，每次读取其中一个。在软件开启的时候，平台会读取软件目录下 WT_SETUP 的所有文件内容，可以通过此接口快速读取其中的内容，也可以按文件中的格式自行添加关键字和数值，使用相应的关键字在软件中读取。只能读取写入 WT_BT_LIMIT、WT_DUT_MIMO、WT_MAC、WT_TESTER、WT_WIFI_LIMIT、WT_ZIGBEE_LIMIT

dutId	SetDutId 接口下传的值
opType	读取或是写入。参考 CallBackFunc.h 中枚举 OperateType
cfgType	操作的文件。参考 CallBackFunc.h 中枚举 CfgType
key	文件中的关键字
value	关键字后的跟着的值
valueSize	读取时 char 数组的大小
elemIndex	值数组中的第几个元素

示例：

```
//------------------------------------------------------------------------------
// 频偏校准设置
//------------------------------------------------------------------------------
WT_FREQ_CAL_DEFAULT_REG         = 69            // 频偏寄存器值默认值.
WT_FREQ_CAL_DEFAULT_REG_5G      = 20            // 5G频偏寄存器值默认值.

//------------------------------------------------------------------------------
// 功率校准设置
//------------------------------------------------------------------------------
//功率寄存器值默认值
//                                [1-5]     [6-10]     [ 11-14]
WT_POW_CAL_DEFAULT_REG_2_4G_CH0     128        128         128
WT_POW_CAL_DEFAULT_REG_2_4G_CH1     193        193         194
```

如 WT_DUT_MIMO 中的关键字 WT_DUT_FREQ_REG_RANGE 使用 CfgElementOperationCB 读取
关键字 WT_POW_CAL_DEFAULT_REG_2_4G_CH0 使用 CfgArrayElementOperationCB 读取