SIMINICS TCSPC DLL 编程指南(2.1.2)

上海星秒光电科技有限公司
(版权所有,翻版必究)

目 录

修订历史

日期	内容	版本
2018 年 3 月 8 日	删除接口 dmtch_Enable	1.2.0
2018 年 3 月 8 日	增加接口 GetActualEvents	1.2.0
2018 年 3 月 8 日	增加接口 GetElapsedTime	1.2.0
2018 年 3 月 8 日	增加接口 SetPulseCycle	1.2.0
2018 年 3 月 8 日	增加接口 GetFilePath	1.2.0
2018 年 3 月 8 日	删除接口 dmtch_Enable	1.2.0
2018 年 3 月 23 日	增加对数据符合工作原理的描述	1.2.1
2018 年 3 月 27 日	更改 DLL 计时方式为 APC 时钟	1.2.1
2018 年 5 月 17 日	增加数据内存接口 SetMemoryDataMode，MemoryDataModeEnabled，GetMemoryData，ReleaseMemoryData	1.2.2
2018 年 5 月 17 日	增加接口 SetMaxFileSize	1.2.2
2018 年 6 月 26 日	增加对接口参数的约定	1.2.3
2018 年 10 月 18 日	修改接口 SetBlockedWindow 的参数范围说明	1.2.4
2019 年 1 月 14 日	增加接口 SetGateHLWidth, SetGateDelay, SetStartImpedence, SetStopImpedence, SetStartEdge, SetStopEdge	1.3.0
2019 年 2 月 21 日	将 SetThreshold 分为 SetStartThreshold 和 SetStopThreshold 两个接口	1.3.1
2019 年 4 月 17 日	增加对 GPS 的支持	1.3.2
2019 年 10 月 25 日	增加接口 EnableT3T1,通过 T3 模拟 T1	1.3.3
2019 年 11 月 15 日	增加对更高分辨率支持，达到 16ps	1.3.4
2020 年 7 月 31 日	增加对多设备的支持	2.0
2021 年 2 月 18 日	增加对采集数据流程的说明	2.1
2021 年 5 月 27 日	修正数据 txt 模式保存时时间解析不争取的问题 去掉不必要接口 SetITTRMaxCount	2.1.1
2022 年 2 月 15 日	对接口的参数进行详细说明	2.1.2

1 概述

本文档描述了 SIMINICS TCSPC DLL(简称 DLL)的功能与使用方法，与 DLL 同时发布。

本 DLL 用于对**上海星秒光电科技有限公司**(SIMINICS Optoelectronics Technology Co.,LTD)出品的**多通道时间相关单光子计数系统**（MultiChannel TCSPC System,简称 TCSPC）的控制、数据采集与数据处理。

本 DLL 不能用于对其他设备或系统的控制，请勿移作他用。

2 系统要求

CPU:

单设备：Intel i5/i7/i9/xeon 双核 2.7G 及其以上

多设备并发：CPU Core 数量要大于拟同时运行设备数量

OS：Windows 10

开发语言：C, C++, C#, Labview

程序类型：GUI 应用，支持 Windows 消息循环

数据传输：USB3.0

驱动：Cypress EZ-USB FX3 SDK 官方驱动

3 DLL 体系结构

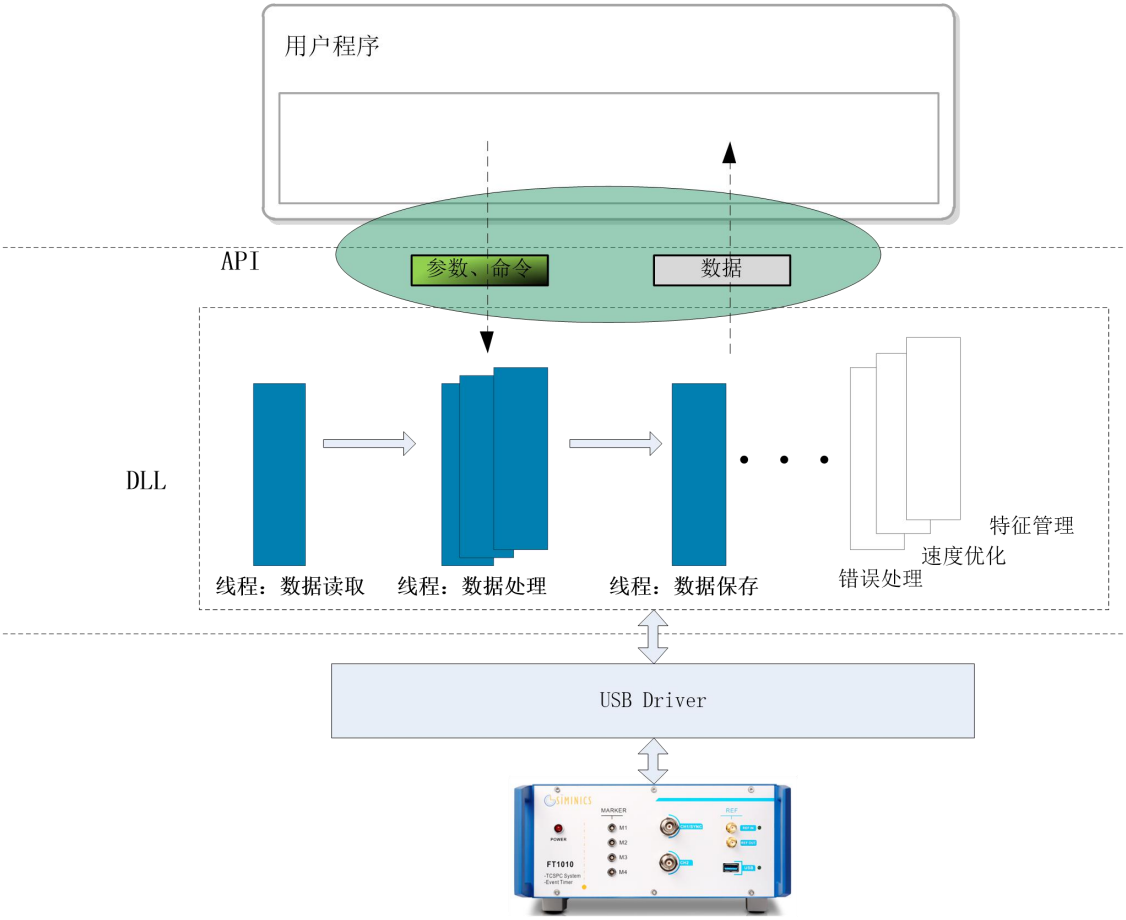

图 1 单设备体系结构

构成 DLL 体系的软硬件组件包括：

1. **多通道时间相关单光子计数系统(TCSPC)**：整个体系的硬件基础，所有的数据都由该设备产生。所有上层的软件结构都是为了更好的为其服务，提高设备速率和效率。
2. **USB3 Driver**：PC 通过 USB3 Driver 与 TCSPC 进行通信。
3. SIMINICS TCSPC DLL：DLL 从其被应用程序加载开始，就运行在多线程状态。主要有 3 个线程：
   1) 数据读取线程：专注读取设备数据，保证数据传输速度。
   2) 数据处理线程：对数据进行解析和用户指定的计算。
   3) 数据保存线程：对原始数据或解析后的数据进行保存，减少磁盘存取速度对前端数据读取和处理造成的影响，提高效率。

   所有线程在用户启动数据读取或数据处理任务前都处于 Idle 状态，不占用系统资源。所有对设备的操作都由底层的多线程为用户执行，用户只需通过 API 设定相关参数，并指定要进行的操作。
   4) 数据符合线程（线程组）：对于每一个符合策略，DLL 都会启动一个相应的线程。符合策略越多，所启动的线程就越多。
4. DLL API：DLL 用户与 API 底层线程交互的接口。
5. 用户程序：DLL 的宿主程序。终端用户通过此程序与 DLL 进行交互，从而实现对 TCSPC 的控制。

DLL 底层使用了 Windows 系统提供的消息钩子接口，用于截获如下消息

1. USB 拔插消息：DLL 需要该消息来获知设备的连接性。
2. 程序关闭消息：DLL 需要该消息来获知用户是否要关闭程序，从而提前释放系统资源。

Windows 系统消息是针对 GUI 应用程序的，本 DLL 对 Windows GUI 程序支持良好。

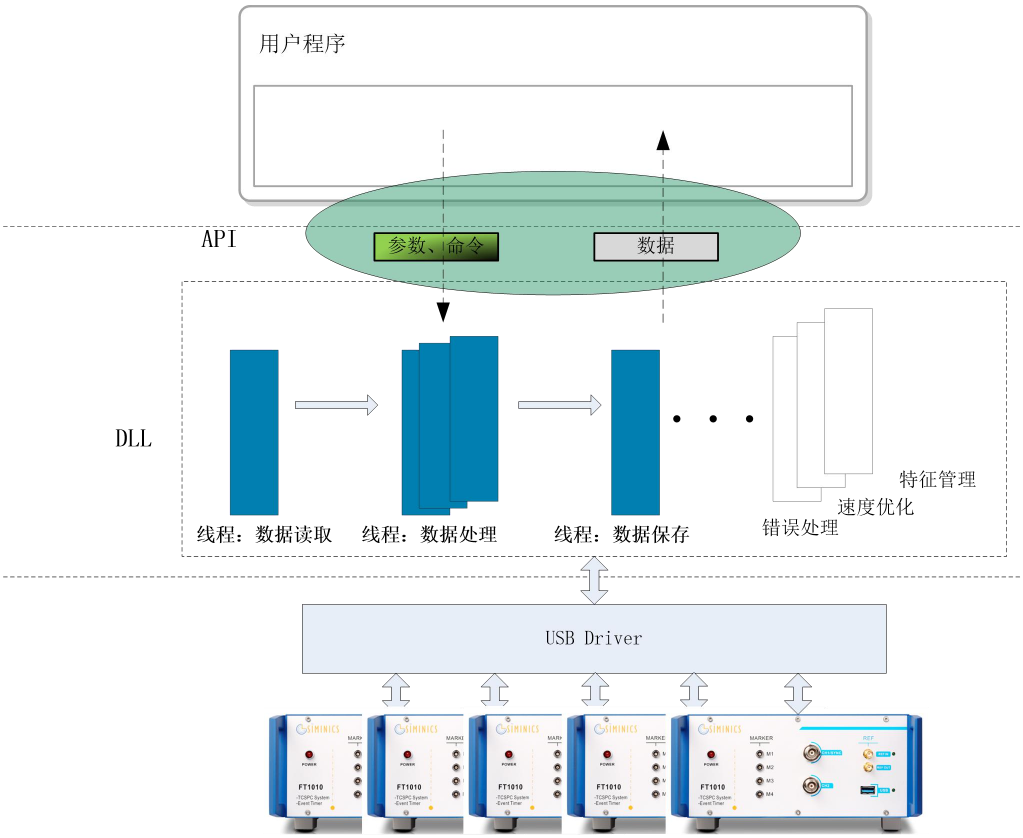

图 2 多设备体系结构

注意：

1. 单设备和多设备使用同一套 DLL。
2. DLL 最多可支持 8 台设备。
3. DLL 同时支持并发模式和非并发模式（普通模式）。

并发模式的启动需要具备两个条件：

1. 设备之间按照并发工作模式接线，其中一台设备为主设备，其他设备为从设备。
2. 使用接口 EnableConcurrentMode 使 DLL 工作在并发模式下。

4 接口说明

4.1 API 列表

GetDevType：	获取设备型号，当前型号有FT1010，FT1020，FT1040，FT1080，MT6420。
GetDllVersion：	获取dll版本号。
GetPartNumber：	获取设备型号编号(Part Number)。
GetSerialNumber：	获取设备序列号(Serial Number)，每个设备拥有不同的Serial Number，单个设备本身和Serial Number一一对应。
SetStartFreqDiv：	设置设备运行参数，Start信号分频。
SetTimeWindowRes：	设置设备运行参数，时间窗口分辨率。
SetBlockedWindow：	设置设备运行参数，屏蔽时间窗口长度。
SetStartThreshold：	设置设备运行参数，Start信号阈值。
SetStopThreshold：	设置设备运行参数，Stop信号阈值。
SetStopDelay：	设置设备运行参数，Stop信号延时。
SetGateHLWidth：	设置设备运行参数，Gate信号脉宽。
SetGateDelay：	设置设备运行参数，Gate信号延时。
SetStartImpedence：	设置设备运行参数，Start信号输入阻抗。
SetStopImpedence：	设置设备运行参数，Stop信号输入阻抗。
SetStartEdge：	设置设备运行参数，Start信号触发方式(上升沿/下降沿)。
SetStopEdge：	设置设备运行参数，Stop信号触发方式(上升沿/下降沿)。
USBConnected：	检查设备与上位机是否通过USB连接。
GetBoardConn：	获取设备每块板卡的连接情况。MT6420拥有多块板卡，此接口获得每块板卡的连接情况。
GetRunMode：	获取设备运行模式，包括T1(柱状图模式)，T2，T3等模式。
GetSignalStatus：	获取设备接入信号状态（信号脉冲统计，包括输入信号和输出信号）。
SetFilePath：	设置DLL所采集的数据文件保存路径。
SetFileMode：	设置DLL所采集的数据文件保存模式，二进制模式或文本模式（解析后）。
GetFilePath：	获取DLL所采集的数据文件的保存路径。
SetMaxFileSize:	设置DLL所采集到的数据文件大小的最大值，超出此值的数据将被保存到多个文件中。
SetMaxBarValue：	设置T1模式(柱状图模式)最大值
SetStatisticsTime：	设置数据采集时间，在T1模式下即T1数据所要采集的总时间，T2，T3，ITTR即对应模式下数据采集的总时间。
SetT1Interval:	设置T1(柱状图模式)模式下数据的刷新时间间隔
EnableT1：	使能T1(柱状图模式)模式，使指定的设备，板卡，和对应通道进入T1模式工作状态。
EnableT3T1：	使能T3T1模式(通过T3数据来模拟T1模式)，使指定设备，板卡，和对应通道进入T3工作状态，并模拟T1功能。
GetT1Data：	获取T1模式(柱状图模式)运行时所采集到的数据。
GetT1Status：	获取T1模式(柱状图模式)运行时的设备状态，包括柱状图最大值及其对应的坐标，以及刷新次数。
EnableITTR:	使能ITTR模式，使指定设备，板卡，和对应通道进入ITTR工作状态。
SetITTRInterval:	设置ITTR数据统计时间间隔。
SetTTTREndMode：	设置TTTR模式(T2，T3模式都属于TTTR)结束方式，是按采集时间停止还是按事件计数停止。
EnableTTTR：	使能TTTR模式，使指定设备，板卡，和对应通道进入ITTR工作状态
GetActualEvents：	获取设备运行时，TTTR数据中的事件计数。
SetMemoryDataMode:	设置内存数据模式，即用户可以不通过文件，直接从内存中同步、实时地获取DLL所采集到的数据。
MemoryDataModeEnabled:	查询内存数据模式是否已经开启
GetMemoryData:	开启内存数据模式后，通过此接口获取数据。
ReleaseMemoryData:	开启内存数据模式后，通过此接口释放获取的数据。
dmtch_Initialize：	数据符合模式，初始化。
dmtch_AddStrategy：	数据符合模式，添加数据符合策略。
dmtch_GetStratgyCount：	数据符合模式，获取数据符合个数。
dmtch_GetChlCount：	数据符合模式，获取通道事件计数。
dmtch_GetChlData：	数据符合模式中，获取数据符合运行时，某个通道的数据。
SetPulseCycle：	信号脉冲统计周期配置。
MarkMainDevice：	多设备工作在并发模式时，标记主设备。
EnableConcurrentMode：	使能并发模式。
StartTask：	启动任务，包括T1，T2，T3，T3T1。
StopTask：	停止正在运行的任务。

IsTaskCompleted:　　查询当前任务是否已完成。
GetElapsedTime:　　查询当前任务已经运行了多久。

4.2 错误定义

DLL 调用过程中发生的错误通过各函数的返回值来通知用户。常用错误定义如下：

```
#define FTMT_ERROR_NONE 0 //无错误
#define FTMT_ERROR_USB_CONN 0x1001 //USB通信错误
#define FTMT_ERROR_SYS_BUSY 0x1002 //系统正忙(任务正在运行中)
#define FTMT_ERROR_PARAM 0x1003 //参数错误(用户提供的参数不符合要求)
#define FTMT_ERROR_MEM_ACCESS 0x1004 //内存访问错误(用户提供的内存地址无法访问)
#define FTMT_ERROR_MEM_LOW 0x1005 //内存不足
#define FTMT_ERROR_FILE_ACCESS 0x1006 //磁盘访问错误(磁盘满或没有访问权限)
#define FTMT_ERROR_CHANNEL_ENABLE 0x1007 //通道使用前没有使能
#define FTMT_ERROR_TTTR_ENABLE 0x1008 //TTTR功能使用前没有使能
#define FTMT_ERROR_BOARD_ENABLE 0x1009 //板卡没有使能
#define FTMT_EEROR_NO_MODE 0x1010 //指定的运行模式不存在
#define FTMT_EEROR_MODE_MATCH 0x1011 //指定的运行模式与实际运行模式不匹配
#define FTMT_ERROR_STRAT_LIMIT 0x1012 //数据符合的策略个数超过所允许的最大值
#define FTMT_ERROR_FATAL 0x1080 //严重错误，DLL需要重新加载或程序重启
```

4.3 常用宏说明

设备型号的宏表示：

```
#define FTMT_DEV_TYPE_FT1010 (0x1010) //设备型号FT1010
#define FTMT_DEV_TYPE_FT1020 (0x1020) //设备型号FT1020
#define FTMT_DEV_TYPE_FT1040 (0x1040) //设备型号FT1040
#define FTMT_DEV_TYPE_FT1080 (0x1080) //设备型号FT1080
#define FTMT_DEV_TYPE_MT6420 (0x6420) //设备型号MT6420
```

Start 信号分频的宏表示：

```
#define FTMT_START_FREQ_DIV_1 (1) //Start信号1分频
#define FTMT_START_FREQ_DIV_2 (2) //Start信号2分频
#define FTMT_START_FREQ_DIV_4 (4) //Start信号4分频
#define FTMT_START_FREQ_DIV_8 (8) //Start信号8分频
```

设备分辨率的宏表示：

```
#define FTMT_WIN_RES_PS_1 (0) //保留定义，但设备暂不支持该分辨率。
#define FTMT_WIN_RES_PS_2 (1) //保留定义，但设备暂不支持该分辨率。
#define FTMT_WIN_RES_PS_4 (2) //保留定义，但设备暂不支持该分辨率。
#define FTMT_WIN_RES_PS_8 (3) //保留定义，但设备暂不支持该分辨率。
#define FTMT_WIN_RES_PS_16 (4) //分辨率，16ps。
#define FTMT_WIN_RES_PS_32 (5) //分辨率，32ps。
#define FTMT_WIN_RES_PS_64 (6) //分辨率，64ps
#define FTMT_WIN_RES_PS_128 (7) //分辨率，128ps
#define FTMT_WIN_RES_PS_256 (8) //分辨率，256ps
#define FTMT_WIN_RES_PS_512 (9) //分辨率，512ps
#define FTMT_WIN_RES_PS_1024 (10)
#define FTMT_WIN_RES_PS_2048 (11)
#define FTMT_WIN_RES_PS_4096 (12)
#define FTMT_WIN_RES_PS_8192 (13)
#define FTMT_WIN_RES_PS_16384 (14)
#define FTMT_WIN_RES_PS_32768 (15)
#define FTMT_WIN_RES_PS_65536 (16)
#define FTMT_WIN_RES_PS_131072 (17)
#define FTMT_WIN_RES_PS_262144 (18)
#define FTMT_WIN_RES_PS_524288 (19)
#define FTMT_WIN_RES_PS_1048576 (20)
#define FTMT_WIN_RES_PS_2097152 (21)
#define FTMT_WIN_RES_PS_4194304 (22)
```

```
#define FTMT_WIN_RES_PS_8388608 (23)
#define FTMT_WIN_RES_PS_16777216 (24)
#define FTMT_WIN_RES_PS_33554432 (25)
```

设备运行模式的宏表示：

```
#define FTMT_TASK_RUN_MODE_T1 (0x01) //设备运行模式，T1
#define FTMT_TASK_RUN_MODE_T2 (0x02) //设备运行模式，T2
#define FTMT_TASK_RUN_MODE_T3 (0x03) //设备运行模式，T3
#define FTMT_TASK_RUN_MODE_CO (0x04) //设备运行模式，数据符合
#define FTMT_TASK_RUN_MODE_T3T1 (0x05) //设备运行模式，T3模拟T1
#define FTMT_TASK_RUN_MODE_ITTR_T (0x06) //设备运行模式，ITTR_T
#define FTMT_TASK_RUN_MODE_ITTR_M (0x07)//设备运行模式，ITTR_M
#define FTMT_TASK_RUN_MODE_IDLE (0x10) //设备运行模式，空闲
#define FTMT_TASK_RUN_MODE_ERR (0x20) //设备运行模式，出错
```

设备数据采集板号的宏表示：

```
#define FTMT_BOARD_A (0) //设备板A
#define FTMT_BOARD_B (1) //设备板B
#define FTMT_BOARD_C (2) //设备板C
#define FTMT_BOARD_D (3) //设备板D
#define FTMT_BOARD_E (4) //设备板E
#define FTMT_BOARD_F (5) //设备板F
#define FTMT_BOARD_G (6) //设备板G
#define FTMT_BOARD_H (7) //设备板H
```

设备通道号的宏表示：

```
#define FTMT_CHANNEL_0 (0) //设备通道号0
#define FTMT_CHANNEL_1 (1) //设备通道号1
#define FTMT_CHANNEL_2 (2) //设备通道号2
#define FTMT_CHANNEL_3 (3) //设备通道号3
#define FTMT_CHANNEL_4 (4) //设备通道号4
#define FTMT_CHANNEL_5 (5) //设备通道号5
#define FTMT_CHANNEL_6 (6) //设备通道号6
#define FTMT_CHANNEL_7 (7) //设备通道号7
```

数据保存模式的宏表示：

```
#define FTMT_FILE_SAVE_MODE_NONE (0) //数据保存模式：不保存
#define FTMT_FILE_SAVE_MODE_BIN (1) //数据保存模式：保存为二进制
#define FTMT_FILE_SAVE_MODE_TEXT (2) //数据保存模式：保存为文本
```

TTTR 数据采集停止模式的宏表示：

```
#define FTMT_TTTR_END_MODE_TIME (0) //TTTR数据采集停止模式：时间停止
#define FTMT_TTTR_END_MODE_EVENT (1) //TTTR数据采集停止模式：事件停止
```

通道掩码的宏表示：

```
#define CH_MASK_0 ((__int8)(0x01)) //通道0掩码
#define CH_MASK_1 ((__int8)(0x01<<FTMT_CHANNEL_1)) //通道1掩码
#define CH_MASK_2 ((__int8)(0x01<<FTMT_CHANNEL_2)) //通道2掩码
#define CH_MASK_3 ((__int8)(0x01<<FTMT_CHANNEL_3)) //通道3掩码
#define CH_MASK_4 ((__int8)(0x01<<FTMT_CHANNEL_4)) //通道4掩码
#define CH_MASK_5 ((__int8)(0x01<<FTMT_CHANNEL_5)) //通道5掩码
#define CH_MASK_6 ((__int8)(0x01<<FTMT_CHANNEL_6)) //通道6掩码
#define CH_MASK_7 ((__int8)(0x01<<FTMT_CHANNEL_7)) //通道7掩码
#define CH_MASK_NONE ((__int8)(0x00)) //清除所有通道的掩码
#define CH_MASK_ALL ((__int8)(0xFF)) //所有通道(0~7)掩码
```

设备数据采集板数量和每个板通道数量的宏表示：

```
#define FT1010_CHANNEL_NUM (1) //FT1010通道数量
#define FT1020_CHANNEL_NUM (2) //FT1020通道数量
#define FT1040_CHANNEL_NUM (4) //FT1040通道数量
#define FT1080_CHANNEL_NUM (8) //FT1080通道数量
```

```
#define MT6420_BOARD_NUM (8) //MT6420数据采集板数量
#define MT6420_CHANNEL_NUM_PER_BOARD (8) //MT6420每个数据采集板的通道数量
```

5 接口详解

用户在调用某些接口时，需要指定 devId 参数，它表示设备编号。

与 PC 主机相连接的第 1 个设备 devId 值为 0，宏定义为：DEV_ID_0，

以此类推，第 2 个设备 devId 值为 1，宏定义为：DEV_ID_1，

第 8 个设备 devId 值为 7，宏定义为：DEV_ID_7。

TDC DLL 当前最多支持 8 个设备。

设备拔出，并再次与 PC 连接后，其当前 devId 与上次可能不同，视当前已连接的设备数量而定。

5.1 设备信息获取

5.1.1 GetDevType

描述：

获取设备型号。

DLL 所支持的型号有：FT1010，FT1020，FT1040，FT1080，MT6420

声明：

```
int GetDevType(int devId, char* devName);
```

参数：

devId：输入参数，设备编号，可能值为：

0，设备 0，对应宏表示为：DEV_ID_0

1，设备 1，对应宏表示为：DEV_ID_1

2，设备 2，对应宏表示为：DEV_ID_2

3，设备 3，对应宏表示为：DEV_ID_3

4，设备 4，对应宏表示为：DEV_ID_4

5，设备 5，对应宏表示为：DEV_ID_5

6，设备 6，对应宏表示为：DEV_ID_6

7，设备 7，对应宏表示为：DEV_ID_7

如果 devId 不存在，则调用出错。

devName: 输出参数，设备型号的字符串表示。用户在调用时，要确保 devName 指向有效的地址；调用结束后，devName 将存储设备型号名称。

如果 devName 为空指针，则调用出错。

如果 devName 指向未定义地址，则会造成不可预料之错误，甚至程序崩溃。

返回值：

设备型号的 16 进制数字表示。可能值及其意义：

0x1010: 设备型号为 FT1010，所对应的宏为 FTMT_DEV_TYPE_FT1010，devName 相应值为："FT1010"。

0x1020: 设备型号为 FT1020，所对应的宏为 FTMT_DEV_TYPE_FT1020，devName 相应值为："FT1020"。

0x1040: 设备型号为 FT1040，所对应的宏为 FTMT_DEV_TYPE_FT1040，devName 相应值为："FT1040"。

0x1080: 设备型号为 FT1080，所对应的宏为 FTMT_DEV_TYPE_FT1080，devName 相应值为："FT1080"。

0x6420: 设备型号为 MT6420，所对应的宏为 FTMT_DEV_TYPE_MT6420，devName 相应值为："MT6420"。

-1: 出错。

示例代码：

```
char d_str[64];
int d_type;
d_type = GetDevType (DEV_ID_0, d_str);
switch(d_type)
{
	case FTMT_DEV_TYPE_FT1010:
	//Code for FT1010
	break;
	case FTMT_DEV_TYPE_FT1020:
	//Code for FT1020
	break;
	case FTMT_DEV_TYPE_FT1040:
	//Code for FT1040
	break;
	case FTMT_DEV_TYPE_FT1080:
	//Code for FT1080
	break;
	case FTMT_DEV_TYPE_MT6420:
	//Code for MT6420
	break;
	default:
	//Code for Impossible
	return;
}
```

```
std::string msg("The Device is:");
msg += d_str;
std::cout << msg;
```

5.1.2 GetSerialNumber

描述：

获取设备序列号。

声明：

```
uint32_t GetSerialNumber(int devId);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

返回值：

32 位，无符号之设备序列号。

示例代码：

```
char d_str[64];
uint32_t serial = GetSerialNumber (DEV_ID_0);
std::string msg("The Serial Number is:");
msg += std::to_string(serial);
std::cout << msg;
```

5.1.3 GetPartNumber

描述：

获取设备部件号。

声明：

```
uint32_t GetPartNumber(int devId);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

返回值：

32 位，无符号之设备序列号。

示例代码：

```
char d_str[64];
uint32_t part = GetPartNumber (DEV_ID_0);
std::string msg("The Part Number is:");
msg += std::to_string(part);
std::cout << msg;
```

5.2 设备参数设置

5.2.1 SetStartFreqDiv

描述：

设置 Start 信号分频。

声明：

```
int SetStartFreqDiv(int devId, int freqDiv);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

freqDiv：输入参数，Start 分频。可接受的值为：1，2，4，8。

1：所对应的宏为 FTMT_START_FREQ_DIV_1。

2：所对应的宏为 FTMT_START_FREQ_DIV_2。

4：所对应的宏为 FTMT_START_FREQ_DIV_4。

8：所对应的宏为 FTMT_START_FREQ_DIV_8。

其他值：调用返回“参数错误”指示。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int f_div = FTMT_START_FREQ_DIV_4;
if(SetStartFreqDiv(DEV_ID_0, f_div)!=  FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.2 SetTimeWindowRes

描述：

设置 T1/T3 时间窗分辨率。

声明：

```
int SetTimeWindowRes(int devId, int winRes);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

winRes：输入参数，分辨率。可接受的值为：0~25。

0：保留，如果配置该值，函数返回 FTMT_ERROR_PARAM。

1：保留，如果配置该值，函数返回 FTMT_ERROR_PARAM。

2：保留，如果配置该值，函数返回 FTMT_ERROR_PARAM。

3：保留，如果配置该值，函数返回 FTMT_ERROR_PARAM。

4：设置分辨率为 16ps，即 2^4ps，所对应的宏为 FTMT_WIN_RES_PS_16。

5：设置分辨率为 32ps，即 2^5ps，所对应的宏为 FTMT_WIN_RES_PS_32。

6：设置分辨率为 64ps，即 2^6ps，所对应的宏为 FTMT_WIN_RES_PS_64。

7：设置分辨率为 128ps，即 2^7ps，所对应的宏为 FTMT_WIN_RES_PS_128。

8：设置分辨率为 256ps，即 2^8ps，所对应的宏为 FTMT_WIN_RES_PS_256。

9：设置分辨率为 512ps，即 2^9ps，所对应的宏为 FTMT_WIN_RES_PS_512。

10：设置分辨率为 1024ps，即 2^10ps，所对应的宏为 FTMT_WIN_RES_PS_1024。

11：设置分辨率为 2048ps，即 2^11ps，所对应的宏为 FTMT_WIN_RES_PS_2048。

12：设置分辨率为 4096ps，即 2^12ps，所对应的宏为 FTMT_WIN_RES_PS_4096。

13：设置分辨率为 8192ps，即 2^13ps，所对应的宏为 FTMT_WIN_RES_PS_8192。

14：设置分辨率为 16384ps，即 2^14ps，所对应的宏为 FTMT_WIN_RES_PS_16384。

15：设置分辨率为 32768ps，即 2^15ps，所对应的宏为 FTMT_WIN_RES_PS_32768。

16：设置分辨率为 65536ps，即 2^16ps，所对应的宏为 FTMT_WIN_RES_PS_65536。

17：设置分辨率为 131072ps，即 2^17ps，所对应的宏为 FTMT_WIN_RES_PS_131072。

18：设置分辨率为 262144ps，即 2^18ps，所对应的宏为 FTMT_WIN_RES_PS_262144。

19：设置分辨率为 524288ps，即 2^19ps，所对应的宏为 FTMT_WIN_RES_PS_524288。

20：设置分辨率为 1048576ps，即 2^20ps，所对应的宏为 FTMT_WIN_RES_PS_1048576。

21：设置分辨率为 2097152ps，即 2^21ps，所对应的宏为 FTMT_WIN_RES_PS_2097152。

22：设置分辨率为 4194304ps，即 2^22ps，所对应的宏为 FTMT_WIN_RES_PS_4194304。

23：设置分辨率为 8388608ps，即 2^23ps，所对应的宏为 FTMT_WIN_RES_PS_8388608。

24：设置分辨率为 16777216ps，即 2^24ps，所对应的宏为 FTMT_WIN_RES_PS_16777216。

25：设置分辨率为 33554432ps，即 2^25ps，所对应的宏为 FTMT_WIN_RES_PS_33554432。

其他值：调用返回“参数错误”指示。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
int w_res = FTMT_WIN_RES_PS_64;
if(SetTimeWindowRes(DEV_ID_0, w_res) !=FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.3 SetBlockedWindow

描述：

设置柱状图(T1)屏蔽时间窗口。

声明：

```
int SetBlockedWindow(int devId, float bWin);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

bWin：输入参数，屏蔽时间窗口长度，单位（ns）。

bWin 的值须介于[0,1024]之间。如果不在该区间，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
float b_win = 180;//ns
if(SetBlockedWindow (DEV_ID_0, b_win) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.4 SetStartThreshold

描述：

设置 start 信号阈值。

声明：

```
int SetStartThreshold(int devId, int brd, int startThrsh);
```

参数：

devId：输入参数，设备编号,具体涵义参见 5.1.1 节。

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。

0：板编号 A，用宏表示为 FTMT_BOARD_A。

1：板编号 B，用宏表示为 FTMT_BOARD_B。

2：板编号 C，用宏表示为 FTMT_BOARD_C。

3：板编号 D，用宏表示为 FTMT_BOARD_D。

4：板编号 E，用宏表示为 FTMT_BOARD_E。

5：板编号 F，用宏表示为 FTMT_BOARD_F。

6：板编号 G，用宏表示为 FTMT_BOARD_G。

7：板编号 H，用宏表示为 FTMT_BOARD_H。

其他值：调用返回“参数错误”FTMT_ERROR_PARAM 指示。

startThrsh：输入参数，Start 信号阈值，单位(mV)。该值须介于[-1250, 0]之间，如果不在允许值范围内，则函数返回 FTMT_ERROR_PARAM。对于负电平设备该值是有效的；对于正电平设备，函数不会报错，但设置无意义。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int brd=FTMT_BOARD_A;
int s_thrsh = -100; //Start阈值100mV
int t_thrsh = -100; //Stop阈值100mV
if(SetStartThreshold(DEV_ID_0, brd, s_thrsh) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.5 SetStopThreshold

描述：

设置 Stop 信号阈值。

声明：

```
int SetStopThreshold(int devId, int brd, int chl, int stopThrsh);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。

0：板编号 A，用宏表示为 FTMT_BOARD_A。

1：板编号 B，用宏表示为 FTMT_BOARD_B。

2：板编号 C，用宏表示为 FTMT_BOARD_C。

3：板编号 D，用宏表示为 FTMT_BOARD_D。

4：板编号 E，用宏表示为 FTMT_BOARD_E。

5：板编号 F，用宏表示为 FTMT_BOARD_F。

6：板编号 G，用宏表示为 FTMT_BOARD_G。

7：板编号 H，用宏表示为 FTMT_BOARD_H。

chl：输入参数，通道编号，可接受值为：0，1，2，3，4，5，6，7。

0：通道编号 0，用宏表示为 FTMT_CHANNEL_0。

1：通道编号 1，用宏表示为 FTMT_CHANNEL_1。

2：通道编号 2，用宏表示为 FTMT_CHANNEL_2。

3：通道编号 3，用宏表示为 FTMT_CHANNEL_3。

4：通道编号 4，用宏表示为 FTMT_CHANNEL_4。

5：通道编号 5，用宏表示为 FTMT_CHANNEL_5。

6：通道编号 6，用宏表示为 FTMT_CHANNEL_6。

7：通道编号 7，用宏表示为 FTMT_CHANNEL_7。

stopThrsh：输入参数，Stop 信号阈值，单位(mV)。该值须介于[-1250, 0]之间，如果不在允许值范围内，则函数返回 FTMT_ERROR_PARAM。对于负电平设备，该值是有效的；对于正电平设备，函数不会报错，但设置无意义。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int brd=FTMT_BOARD_A;
int chl=FTMT_CHANNEL_0;
int t_thrsh = -100; //Stop阈值100mV
if(SetStopThreshold(DEV_ID_0, brd, chl, t_thrsh) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

说明：

现阶段，对于不同型号设备，只支持对第一板卡的第一通道进行 Stop 阈值设置。

5.2.6 SetStopDelay

描述：

设置 Stop 信号延时。

声明：

```
int SetStopDelay(int devId, int brd, int chl, float delay);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。其意义请参见 4.3 节。

chl：输入参数，通道编号，可接受值为：0，1，2，3，4，5，6，7。

0：通道编号 0，用宏表示为 FTMT_CHANNEL_0。

1：通道编号 1，用宏表示为 FTMT_CHANNEL_1。

2：通道编号 2，用宏表示为 FTMT_CHANNEL_2。

3：通道编号 3，用宏表示为 FTMT_CHANNEL_3。

4：通道编号 4，用宏表示为 FTMT_CHANNEL_4。

5：通道编号 5，用宏表示为 FTMT_CHANNEL_5。

6：通道编号 6，用宏表示为 FTMT_CHANNEL_6。

7：通道编号 7，用宏表示为 FTMT_CHANNEL_7。

delay：输入参数，Stop 信号延时，单位(ns)。该值须介于[0, 1288]之间，如果设置值不在该区间内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
int brd= FTMT_BOARD_A;
int chl = FTMT_CHANNEL_1; //Start阈值100mV
float t_dlys[] = {0.0f, 1.0f, 32.0f, 10.5f, 6.4f, 9.8f, 3.0f, 5.3f};
for(int i=0; i < sizeof(t_dlys)/sizeof(float); i++)
{
	if(SetStopDelay(DEV_ID_0, brd, i, t_dlys[i]) != FTMT_ERROR_NONE)
	{
		//错误处理代码
	}
}
```

5.2.7 SetPulseCycle

描述：

设置 Start/Stop 信号脉冲统计周期，主要用于在不同统计周期下监测信号强度。

声明：

```
int SetPulseCycle(int devId, int brd, int cycle);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，所要设置的板号。

cycle：输入参数，Start/Stop 信号脉冲统计周期，单位(ms)。该值须介于[100, 1024]之间，如果设置值不在该区间内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
int brd= 0;
int cycle = 100;
if(SetPulseCycle (DEV_ID_0, brd, cycle) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.8 SetGateHLWidth

描述：

设置 Gate 信号脉宽配置。

声明：

```
int SetGateHLWidth(int devId, int width);
```

参数：

devId：输入参数，设备编号。

width：输入参数，Gate 信号脉宽，单位(ns)。该值须介于[1, 0xFFFF]之间，如果设置值不在该区间内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
int gate_width = 5;
if(SetGateHLWidth (DEV_ID_0, width) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.9 SetGateDelay

描述：

设置 Gate 信号延迟配置。

声明：

```
int SetGateDelay (int devId, int delay);
```

参数：

devId：输入参数，设备编号。

delay：输入参数，Gate 信号延时，单位(ns)。该值须介于[1, 0xFFFF]之间，如果设置值不在该区间内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int gate_delay = 300;
if(SetGateDelay (DEV_ID_0, gate_delay) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.10 SetStartImpedence

描述：

设置 Start 信号输入阻抗配置，只支持 FT10X0 设备。

声明：

```
int SetStartImpedence (int devId, int impedence);
```

参数：

devId：输入参数，设备编号。

impedence：输入参数，Start 信号输入阻抗选择。该值只能是 0 或 1，可用宏表示：

```
#define FT10X0_INPUT_IMPEDENCE_HIGH (0) //Start信号输入高阻抗
#define FT10X0_INPUT_IMPEDENCE_50 (1) //Start信号输入阻抗50欧姆
```

如果设置值不正确，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int FT10X0_INPUT_IMPEDENCE_50 = 0;
if(SetStartImpedence (DEV_ID_0, start_impedence) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.11 SetStopImpedence

描述：

设置 Stop 信号输入阻抗配置，只支持 FT10X0 设备。

声明：

```
int SetStopImpedence (int devId, int brd, int chl,  int impedence);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，板号，只能为 0，表示 FT10X0 设备。

chl：输入参数，通道编号，可接受值为：0，1，2，3，4，5，6，7。

0：通道编号 0，用宏表示为 FTMT_CHANNEL_0。

1：通道编号 1，用宏表示为 FTMT_CHANNEL_1。

2：通道编号 2，用宏表示为 FTMT_CHANNEL_2。

3：通道编号 3，用宏表示为 FTMT_CHANNEL_3。

4：通道编号 4，用宏表示为 FTMT_CHANNEL_4。

5：通道编号 5，用宏表示为 FTMT_CHANNEL_5。

6：通道编号 6，用宏表示为 FTMT_CHANNEL_6。

7：通道编号 7，用宏表示为 FTMT_CHANNEL_7。

impedence：输入参数，Start 信号输入阻抗选择。该值只能是 0 或 1，可用宏表示：

```
#define FT10X0_INPUT_IMPEDENCE_HIGH (0) //Start信号输入高阻抗
#define FT10X0_INPUT_IMPEDENCE_50 (1) //Start信号输入阻抗50欧姆
```

如果设置值不正确，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
int brd = 0;
int chl = 0;
if(SetStopImpedence (DEV_ID_0, brd, chl, FT10X0_INPUT_IMPEDENCE_HIGH) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.12 SetStartEdge

描述：

设置 Start 信号触发方式选择，只支持 FT10X0 设备。

声明：

```
int SetStartEdge (int devId, int edge);
```

参数：

devId：输入参数，设备编号。

edge：输入参数，输入信号触发方式。该值只能是 0 或 1，可用宏表示：

```
#define FT10X0_INPUT_EDGE_RISE (0) //Start信号上升沿触发
#define FT10X0_INPUT_EDGE_FALL (1) //Start信号下降沿触发
```

如果设置值不正确，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
if(SetStartEdge (DEV_ID_0, FT10X0_INPUT_EDGE_RISE) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.2.13 SetStopEdge

描述：

设置 Stop 触发方式选择，只支持 FT10X0 设备。

声明：

```
int SetStopEdge (int devId, int brd, int chl, int edge);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，板号，只能为 0，表示 FT10X0 设备。

chl：输入参数，通道编号，可接受值为：0，1，2，3，4，5，6，7。

0：通道编号 0，用宏表示为 FTMT_CHANNEL_0。

1：通道编号 1，用宏表示为 FTMT_CHANNEL_1。

2：通道编号 2，用宏表示为 FTMT_CHANNEL_2。

3：通道编号 3，用宏表示为 FTMT_CHANNEL_3。

4：通道编号 4，用宏表示为 FTMT_CHANNEL_4。

5：通道编号 5，用宏表示为 FTMT_CHANNEL_5。

6：通道编号 6，用宏表示为 FTMT_CHANNEL_6。

7：通道编号 7，用宏表示为 FTMT_CHANNEL_7。

impedence：输入参数，Start 信号输入阻抗选择。该值只能是 0 或 1，可用宏表示：

```
#define FT10X0_INPUT_EDGE_RISE (0) //Start信号上升沿触发
#define FT10X0_INPUT_EDGE_FALL (1) //Start信号下降沿触发
```

如果设置值不正确，则函数返回 FTMT_ERROR_PARAM。

示例代码：

```
if(SetStopEdge (DEV_ID_0, FT10X0_INPUT_EDGE_FALL) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.3 设备状态获取

5.3.1 USBConnected

描述：

检测 USB 是否连通。

声明：

```
bool USBConnected(int devId);
```

参数：

devId: 所要检查的设备号。

返回值：

USB 是否已经连通。可能值及其意义：

true: USB 已连通。

false：USB 未连通。

示例代码：

```
if(!USBConnected(DEV_ID_0))
{
    //错误处理代码
}
```

5.3.2 GetRunMode

描述：

获取当前运行模式。

声明：

```
int GetRunMode(char* rMode);
```

参数：

rMode：输出参数，设备运行模式的字符串表示。

返回值：

设备运行模式的数字表示。可能的值为：

0x1：设备运行在 T1 模式，用宏表示为 FTMT_TASK_RUN_MODE_T1，rMode 相应返回值为“T1”。

0x2：设备运行在 T2 模式，用宏表示为 FTMT_TASK_RUN_MODE_T2，rMode 相应返回值为“T2”。

0x3：设备运行在 T3 模式，用宏表示为 FTMT_TASK_RUN_MODE_T3，rMode 相应返回值为“T3”。

0x4：设备运行在数据符合模式，用宏表示为 FTMT_TASK_RUN_MODE_CO，rMode 相应返回值为“Correlation”。

0x5：设备运行在 T3T1 模式（通过 T3 模拟 T1），用宏表示为 FTMT_TASK_RUN_MODE_T3T1，rMode 相应返回值为“T3T1”。

0x10：设备空闲，用宏表示为 FTMT_TASK_RUN_MODE_IDLE，rMode 相应返回值为“Idle”。

0x20：设备状态出错，用宏表示为 FTMT_TASK_RUN_MODE_ERR，rMode 相应返回值为“Error”。

示例代码：

```
char s_mod[64];
int i_mode = GetRunMode (s_mode);
switch(i_mode)
{
    case FTMT_TASK_RUN_MODE_T1:
    //Code for T1
    break;
    case FTMT_TASK_RUN_MODE_T2:
    //Code for T2
    break;
    case FTMT_TASK_RUN_MODE_T3:
    //Code for T3
    break;
    case FTMT_TASK_RUN_MODE_CO:
    //Code for Data Correlation
    break;
    case FTMT_TASK_RUN_MODE_T3T1:
    //Code for T3T1
    break;
    case FTMT_TASK_RUN_MODE_IDLE:
    //Code for Idle
    break;
    default:
    //Code for Error
    return;
}
String msg(“The Device is Running With Mode:”);
msg += s_mod;
MessageBox::Show(msg);
```

5.3.3 GetSignalStatus

描述：

获取接入设备的信号状态（信号脉冲统计）。

声明：

```
int GetSignalStatus(int devId, int brd, int chl, unsigned__int32* strtCnt, unsigned __int32* stopCnt);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

chl：输入参数，通道编号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见

4.3 节。

strtCnt：输出参数，信号脉冲统计。

stopCnt：输出参数，Stop 信号脉冲统计。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
char s_mod[64];
int brd= FTMT_BOARD_A;
int chl = FTMT_CHANNEL_0;
unsigned int s_cnt; //Start脉冲统计
unsigned int t_cnt; //Stop脉冲统计
if(GetSignalStatus(DEV_ID_0, brd, chl, &s_cnt, &t_cnt) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
//数据处理代码
```

5.4 软件行为

5.4.1 GetDllVersion

描述：

获得 DLL 版本，客户如果对系统升级可能使用到。

声明：

```
bool GetDllVersion(char* version);
```

参数：

version：输出参数，返回当前 DLL 版本号，格式为 mainVer.subVer.fixVer_YMD，如 1.3.2_20190313。

设置成功或者失败。可能值及其意义：

true: 成功。

false: 失败。

示例代码：

```
char* p_save = "E:\\Siminics\Data";
if(SetPath(1, p_save) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.4.2 SetFilePath

描述：

设置采集数据文件保存路径。

DLL 在运行过程中，会按照客户要求（通过 API 指定），将数据进行保存。其数据保存的位置就通过 SetFilePath 指定。

如果用户不指定路径，并且要求保存数据，则数据采集功能不能正确启动。

如果用户指定的路径不存在，DLL 在开始数据采集时会自动创建路径。

如果用户指定的路径盘符（如 D:\, E:\）不存在，并要求保存数据，则数据采集功能不能正确启动。

声明：

```
int SetFilePath(int devId, int rMode,char* path);
```

参数：

devId：输入参数，设备编号。

rMode：输入参数，DLL 运行模式。可接受值为 1，2，3，4，5。各值具体意义请参见 4.3 节。

path：输入参数，数据保存路径，C 字符串类型，以’\0’为结束标志。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
if(SetFilePath(FTMT_TASK_RUN_MODE_T1, "E:\\Siminics\\data") != FTMT_ERROR_NONE)
{
    std::cout << “SetFilePath Error!”;
}
```

5.4.3 GetFilePath

描述：

获得当前数据文件保存路径。

DLL 在运行过程中，会按照客户指定的路径，将数据保存在文件中。用户可以通过此接口获得具体的文件保存路径。

声明：

```
int   GetFilePath(int devId , int rMode,   char* path);
```

参数：

devId：输入参数，设备编号。

rMode：输入参数，DLL 运行模式。可接受值为 1，2，3，4，5。各值具体意义请参见 4.3 节。

path：输出参数，数据保存路径，C 字符串类型，以’\0’为结束标志。必须提供一内存地址，以便接收该文件位置。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
char p_path[256];
if(GetFilePath(DEV_ID_0, 1, p_path) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.4.4 SetFileMode

描述：

设置文件保存方式。

声明：

```
int SetFileMode(int rMode, int sMode, int fMode);
```

参数：

rMode：输入参数，设备运行模式。可接受值为：1，2，3，4，5。各值具体意义请参见 4.3 节。

sMode：输入参数，文件保存格式。可能值为：0，1，2。

0：不保存，用宏表示为 FTMT_FILE_SAVE_MODE_NONE。

1：保存二进制数据，用宏表示为 FTMT_FILE_SAVE_MODE_BIN。

2：保存解析后的文本数据，用宏表示为 FTMT_FILE_SAVE_MODE_TEXT。

在采集 T2/T3 数据时，建议只保存二进制数据，否则采集速度会受到影响，造成设备反压，甚至数据丢失的情况。

fMode：输入参数，数据结果连续保存模式，只对 T1 有效。可能值为：0，1。

0：保存直方图的每次更新，用宏表示为 FTMT_FILE_SERIES_MODE_EACH。

1：保存直方图的最后一次更新，用宏表示为 FTMT_FILE_SERIES_MODE_LAST。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 要求在T1模式运行时，保存解析后数据，并且只保存直方图最后一次更新
*/
int r_mod = FTMT_TASK_RUN_MODE_T1;//T1
int s_mod = FTMT_FILE_SAVE_MODE_TEXT;//保存文本
int f_mod = FTMT_FILE_SERIES_MODE_LAST;//保存直方图最后一次更新
if(SetFileMode(r_mod, s_mod, f_mod) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 要求在T2模式运行时，保存二进制数据
*/
int r_mod = FTMT_TASK_RUN_MODE_T1;//T2
int s_mod = FTMT_FILE_SAVE_MODE_BIN;//保存二进制
int f_mod = 0;//无意义
if(SetFileMode(r_mod, s_mod, f_mod) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.4.5 SetMaxFileSize

描述：

设置数据文件大小所允许的最大值。当数据量超过该值时，DLL 会创建新文件，将数据保存在新文件中。

声明：

```
int SetMaxFileSize(int size);
```

参数：

size：输入参数，文件所允许大小的最大值，单位：M Bytes。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 设置TTTR模式下，文件最大值为500M
*/

Int fSize = 500;
if(SetMaxFileSize (fSize) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5 T1 功能接口

5.5.1 SetMaxBarValue

描述：

指定 T1 模式(柱状图模式)时，柱状图最大值。当实际统计值大于等于设置值时，统计停止，T1 模式结束。

声明：

```
int SetMaxBarValue(unsigned __int64 val);
```

参数：

val：输入参数，所要设置的柱状图最大值。该值所允许的范围为[0, 0xFFFFFFFF]，如果设置值不在该区间范围内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 要求在T1模式运行时，柱状图最大值为1000000
*/
unsigned __int64 m_val = 1000000;//T2
if(SetMaxBarValue (m_val) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.2 SetStatisticsTime

描述：

设置数据采集时间。当实际统计时间大于设置时间时，数据采集结束。

声明：

```
int SetStatisticsTime(int rMode, float sTime);
```

参数：

rMode：输入参数，设备运行模式。可接受值为：1，2，3，4，5。各值具体意义请参见 4.3 节。

sTime：输入参数，数据采集时间，单位：秒(s)。最小值为 0.1s，最大值为 100 * 60 * 60 * 1.0f（1 天）。如果设置值不在该区间范围内，则函数返回 FTMT_ERROR_PARAM。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 设置T2统计时间为10s
*/
float s_tm = 10.0f; //统计时间10s
if(SetStatisticsTime(FTMT_TASK_RUN_MODE_T2，s_tm) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.3 SetT1Interval

描述：

设置 T1 刷新时间间隔。

声明：

```
int SetT1Interval (float sIntvl);
```

参数：

sIntvl：输入参数，柱状图刷新间隔,单位：秒(s)。该参数允许的最小值为 0.01s，如果设置值小于 0.01，则函数会自动将其设为 0.01。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
float i_tm = 0.1f;   //刷新间隔0.1s
if(SetT1Interval (i_tm) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.4 EnableT1

描述：

指定 T1 模式下所需作统计的通道号掩码。

声明：

```
int EnableT1(int devId, int brd, unsigned __int8 mask8);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，所需统计的板号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

mask8：所指定板的通道号掩码。可以用宏来简化掩码设置，宏的具体意义如下：

CH_MASK_0：通道 0 使能掩码。

CH_MASK_1：通道 1 使能掩码。

CH_MASK_2：通道 2 使能掩码。

CH_MASK_3：通道 3 使能掩码。

CH_MASK_4：通道 4 使能掩码。

CH_MASK_5：通道 5 使能掩码。

CH_MASK_6：通道 6 使能掩码。

CH_MASK_7：通道 7 使能掩码。

CH_MASK_NONE：所有通道去使能掩码。

CH_MASK_ALL：所有通道使能掩码。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 对板0的通道0进行T1统计
*/
int brd = int brd= FTMT_BOARD_A;
int chl = FTMT_CHANNEL_0;

unsigned __int8 mask8 |= CH_MASK_0; //通道0
if(EnableT1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 对板0的通道2进行T1统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_2; //通道2
if(EnableT1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 对板0的通道3和通道7进行T1统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_3 | CH_MASK_7; //通道3和通道7
if(EnableT1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.5 EnableT3T1

描述：

指定 T3T1 模式下所需作统计的通道号掩码。

声明：

```
int EnableT3T1(int devId, int brd, unsigned __int8 mask8);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，所需统计的板号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

mask8：所指定板的通道号掩码。可以用宏来简化掩码设置，宏的具体意义如下：

CH_MASK_0：通道 0 使能掩码。

CH_MASK_1：通道 1 使能掩码。

CH_MASK_2：通道 2 使能掩码。

CH_MASK_3：通道 3 使能掩码。

CH_MASK_4：通道 4 使能掩码。

CH_MASK_5：通道 5 使能掩码。

CH_MASK_6：通道 6 使能掩码。

CH_MASK_7：通道 7 使能掩码。

CH_MASK_NONE：所有通道去使能掩码。

CH_MASK_ALL：所有通道使能掩码。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 对板0的通道0进行T1统计
*/
int brd = int brd= FTMT_BOARD_A;
```

```
int chl = FTMT_CHANNEL_0;

unsigned __int8 mask8 |= CH_MASK_0; //通道0
if(EnableT3T1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 对板0的通道2进行T1统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_2; //通道2
if(EnableT3T1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 对板0的通道3和通道7进行T1统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_3 | CH_MASK_7; //通道3和通道7
if(EnableT3T1(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.6 GetT1Data

描述：

获得 T1 模式采集到的数据。

声明：

```
int GetT1Data(int brd, int chl, unsigned __int64 data[FTMT_T1_DATA_LEN] );
```

参数：

data[FTMT_T1_DATA_LEN]：输出参数，T1 柱状图统计数据。每次返回数据长度为固定值 65536，用宏表示为：FTMT_T1_DATA_LEN。该函数不阻塞。用户每次调用都会获得当前统计数据。数据更新时间间隔由接口 SetT1Interval 设置。

brd：输入参数，所需统计的板号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

chl：输入参数，所需统计的通道号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

返回值：

数据获取成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 对板0的通道0进行T1统计
*/
const int v_len = 65536;
unsigned __int64 b_vals[v_len];
int brd = FTMT_BOARD_A;
int chl = FTMT_CHANNEL_0;
if(GetT1Data(b_vals, brd, chl) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.5.7 GetT1Status

描述：

设备在 T1 模式运行时，获取 T1 统计数据状态。

声明：

```
void  GetT1Status(unsigned __int32* xMax, unsigned __int64* yMmax, unsigned __int32* reCnt);
```

参数：

xMax：输出参数，柱状图最大值所对应的 x 坐标。

yMmax：输出参数，柱状图最大值。

reCnt：输出参数，柱状图数据刷新次数。

返回值：

无。

示例代码：

```
unsigned int x_m;
unsigned __int64 y_m;
unsigned int r_cnt;
GetT1Status(&x_m, &y_m, &r_cnt);
printf("Max Value %ull@%d. Refresh Count:%d\n", x_m, y_m, r_cnt);
```

5.5.8 GetElapsedTime

描述：

返回设备在某种模式下，已经运行了多长时间。

声明：

```
float GetElapsedTime(int rMode);
```

参数：

rMode：输入参数，设备运行模式。可接受值为：1，2，3，4，5。各值具体意义请参见 4.3 节。

返回值：

时间长度，单位：秒(s)。

示例代码：

```
float elapsed_time = GetElapsedTime(int rMode);
```

5.6 ITTR 接口功能

5.6.1 EnableITTR

描述：

使能 ITTR 模式。

声明：

```
int EnableITTR (int devId, int brd, uint8_t mask8);
```

参数：

devId：输入参数，设备编号

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。其意义请参见 4.3 节。

mask8：输入参数，所指定板号的通道号掩码。掩码设置可用宏表示，具体方法请参见 4.3 节。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 对板0的通道0进行ITTR统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_0; //通道0
if(EnableITTR(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
```

```
    //错误处理代码
}
```

5.6.2 SetITTRInterval

描述：

设置 ITTR 统计时间间隔。

声明：

```
int SetITTRInterval(int devId, int brd, int intvl);
```

参数：

devId：输入参数，设备编号

brd：输入参数，板编号，可接受值为：0，1，2，3，4，5，6，7。其意义请参见 4.3 节。

intvl：输入参数，ITTR 统计时间间隔，单位 us。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 设置ITTR统计时间间隔为300us
*/
int brd = FTMT_BOARD_A;
if(SetITTRInterval (DEV_ID_0, brd, 300) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.6.3 SetITTREndMode

描述：

设置 ITTR 结束模式。

声明：

```
int SetITTREndMode(int eMode, unsigned __int64 events);
```

参数：

dMode：输入参数，ITTR 结束模式。

0：按时间结束。

1：按事件结束。

events：输入参数，总统计事件数。如选择按时间结束，则该参数忽略。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 设置ITTR按时间结束
*/
if(SetITTREndMode(0, 0) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.7 TTTR 功能接口

5.7.1 SetTTTREndMode

描述：

设置 TTTR 模式统计停止方式。

声明：

```
int SetTTTREndMode(int eMode, unsigned __int64 events); //0:time, 1:events
```

参数：

eMode：输入参数，TTTR 模式停止方式。可接受的值为 0，1。

0：时间停止，用宏表示为 FTMT_TTTR_END_MODE_TIME。

1：事件停止，用宏表示为 FTMT_TTTR_END_MODE_EVENT。

events：输入参数，事件停止方式下，需要统计的最大事件数。如果第一个参数 eMode 为 FTMT_TTTR_END_MODE_TIME，则此参数值忽略。

返回值：

设置成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* TTTR统计时间停止
*/
int e_m = FTMT_TTTR_END_MODE_TIME;
unsigned __int64 e_nums = 0;
if(SetTTTREndMode(e_m, e_nums) != FTMT_ERROR_NONE)
{
    //错误处理代码
}


/*
* TTTR统计事件停止，所统计事件数位1000
*/
int e_m = FTMT_TTTR_END_MODE_EVENT;
unsigned __int64 e_nums = 1000;
if(SetTTTREndMode(e_m, e_nums) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.7.2 GetActualEvents

描述：

当设备工作在 TTTR 模式时，可通过此接口获得 T2、T3 当前统计的实际事件数。

声明：

```
unsigned __int64 GetActualEvents(int rMode);
```

参数：

rMode：输入参数，TTTR 的模式。可接受的值为 2、3，分别表示 T2，T3 模式。

2：工作在 T2 模式。

3：工作在 T3 模式。

返回值：

T2、T3 当前统计的实际事件数。

示例代码：

```
/*
* TTTR统计事件停止，所统计事件数位1000
*/
int e_m = FTMT_TTTR_END_MODE_EVENT;
unsigned __int64 e_nums = 1000;
unsigned __int64 c_nums ;
```

```
if(SetTTTREndMode(e_m, e_nums) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
/*
* 启动设备进行T2数据采集
*/
if(StartTask(FTMT_TASK_RUN_MODE_T2) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

c_nums = GetActualEvents(FTMT_TASK_RUN_MODE_T2)
{
    //错误处理代码
}
```

5.7.3 EnableTTTR

描述：

使能需要运行在 TTTR 模式下的通道。

声明：

```
int EnableTTTR(int devId, int brd, unsigned __int8 mask8);
```

参数：

devId：输入参数，设备编号。

brd：输入参数，所需统计的板号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3

mask8：输入参数，所指定板号的通道号掩码。掩码设置可用宏表示，具体方法请参见 4.3 节。对于 FT1010，在 T2 模式下运行时，固定配置通道 0 和通道 1 使能。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 对板0的通道0进行TTTR统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_0; //通道0
if(EnableTTTR(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

```
/*
* 对板0的通道2进行TTTR统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_2; //通道2
if(EnableTTTR(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}

/*
* 对板0的通道3和通道7进行TTTR统计
*/
int brd = FTMT_BOARD_A;
unsigned __int8 mask8 |= CH_MASK_3 | CH_MASK_7; //通道3和通道7
if(EnableTTTR(DEV_ID_0, brd, mask8) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.7.4 SetMemoryDataMode

描述：

设置内存数据模式。用户可以通过该接口，开启内存数据模式。在该模式下，DLL 将为 TTTR 数据开辟一段缓存，所有的 TTTR 数据都将保存在该缓存内。

当用户通过 GetMemoryData 接口获取数据时，相应数据会从缓存中移除。

当缓存满，用户仍然没有通过 GetMemoryData 移除先前的数据时，缓存中最老的数据将被覆盖。

数据内存模下，DLL 会多拷贝一份数据到缓存，造成软件性能下降。默认不开启。

声明：

```
int SetMemoryDataMode(bool en)
```

参数：

en：输入参数，是否开启内存数据模式。

true：开启数据内存模式。

false：关闭数据内存模式。

返回值：

设置成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 开启数据内存模式
*/
if(SetMemoryDataMode (true) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.7.5 MemoryDataModeEnabled

描述：

查看内存数据模式是否已开启。

声明：

```
bool MemoryDataModeEnabled( );
```

参数：

无。

返回值：

内存数据模式是否已经开启。可能值及其意义：

true: 数据内存模式已经开启。

false: 数据内存模式已经关闭。

示例代码：

```
/*
* 查看内存数据模式是否已经开启
*/
if(!MemoryDataModeEnabled ( ))
{
    //错误处理代码
}
```

5.7.6 GetMemoryData

描述：

在内存数据模式下，从缓存中获得数据。

声明：

```
FtmtRawData* GetMemoryData();
```

参数：

无。

返回值：

TTTR 数据。如果返回值为 nullptr(c++)或 NULL(C)，则表示缓存中已经没有数据。

示例代码：

```
FtmtRawData* dataPtr = GetMemoryData();

if(dataPtr != nullptr)
{
    //数据处理代码
}
```

5.7.7 ReleaseMemoryData

描述：

释放通过 GetMemoryData 接口获得数据所占据的内存。

声明：

```
void ReleaseMemoryData(FtmtRawData* dataPtr);
```

参数：

dataPtr：输入参数，指向 TTTR 数据的指针。

返回值：

无。

示例代码：

```
FtmtRawData* dataPtr = GetMemoryData();

if(dataPtr != nullptr)
{
    //数据处理代码

    //释放内存
    ReleaseMemoryData(dataPtr);
}
```

5.8 数据符合接口

5.8.1 dmtch_Initialize

描述：

初始化数据符合功能。

声明：

```
int dmtch_Initialize();
```

参数：

无。

返回值：

返回值：

操作成功或者失败。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
if (dmtch_Initialize()!= FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.8.2 dmtch_AddStrategy

描述：

添加数据符合策略。

声明：

```
int dmtch_AddStrategy(int idx, int str[], int num, int strSave, int winLen);
```

参数：

idx：输入参数，所要添加策略编号。由用户指定，有效编号值从 0 开始。

str[]：输入参数，所要添加策略的通道号组合。

num：输入参数，所要添加策略的通道号个数。

strSave：输入参数，在进行数据采集时，是否保存符合的数据。

winLen：输入参数,符合运算时的窗口宽度，即 bin 的个数。

返回值：

添加成功或者失败。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
int s_idx = 1;
char* s_name = "Strategy_1";
int w_len = 100;
int strat[] = {0, 3, 5};
if(dmtch_AddStrategy(s_idx, s_name, strat, sizeof(strat)/sizeof(int), true, w_len) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

说明：

DLL 现在可以支持 64 组的数据符合策略。目前不支持板卡之间和设备之间的数据符合。

5.8.3 dmtch_GetStratgyCount

描述：

获得数据符合策略所记录的符合数据数目。

数据符合是一个动态过程。返回的数目随设备运行时间的变化而变化。

声明：

```
unsigned __int64  dmtch_GetStratgyCount(int idx);
```

参数：

idx：输入参数，用户所添加策略编号。通过接口 dmtch_AddStrategy 由用户指定。

返回值：

满足策略要求的符合数据数目。

示例代码：

```
/*
* 获取策略0的符合数目
*/
int s_idx = 1;
int w_len = 100;
int strat[] = {0, 3, 5};
unsigned __int64 count = dmch_GetStrategyCount(0);
//数据处理
```

5.8.4 dmtch_GetChlCount

描述：

获取参与数据符合的各个通道的事件数。

数据符合是一个动态过程。返回的数目随时间的变化而变化。

声明：

```
int dmtch_GetChlCount(unsigned __int64 chlCnt[]);
```

参数：

chlCnt：输出参数，各个通道事件计数保存在该数组中。

返回值：

数据获取成功或失败。

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
const int chl_num = 8;
unsigned __int64 c_cnt[chl_num];
if(!dmtch_GetChlCount(c_cnt) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
for(int i=0; i<chl_num; i++)
{
    printf("Data Item Number for Chanel %d:%ull\n", i, c_cnt[i]);
}
```

5.8.5 dmtch_GetChlData

描述：

获取参与数据符合的某个通道柱状图数据。

声明：

```
int dmtch_GetChlData(int brd, int chl, unsigned __int64 data[FTMT_DMTCH_BAR_LEN]);
```

参数：

brd：输入参数，板号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

chl：输入参数，通道号，可接受值为：0，1，2，3，4，5，6，7。各值具体意义请参见 4.3 节。

data：输出参数，获得的柱状图保存在该数组中。

返回值：

数据获取成功或失败。

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
const int chl_num = 8;
int brd = 0;
int chl = 1;
unsigned __int64 c_cnt[chl_num];
if(!dmtch_GetChData(brd, chl, c_cnt) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
//数据处理代码
```

5.9 并发模式接口

5.9.1 MarkMainDevice

描述：

并发模式中，标记主设备。

声明：

```
int MarkMainDevice(int devId);
```

参数：

devId：输入参数，设备编号。

返回值：

启动是否成功。可能值及其意义：

0：成功，用宏 FTMT_ERROR_NONE 表示。

其他：参见 4.2 节。

示例代码：

```
/*
* 标记设备0为主设备
*/
if(MarkMainDevice (DEV_ID_0) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.9.2 EnableConcurrentMode

描述：

开启或关闭并发模式。

声明：

```
int EnableConcurrentMode (bool en);
```

参数：

en：是否使能并发模式。

返回值：

启动是否成功。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 使能并发模式
*/
if(EnableConcurrentMode(true) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.10 任务管理

5.10.1 StartTask

描述：

启动设备进行数据采集。

声明：

```
int StartTask(int rMode);
```

参数：

rMode：输入参数，设备将要工作的模式。可接受值为：1，2，3，4，5。各值具体意义请参见 4.3 节。

返回值：

启动是否成功。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 启动设备进行T1数据采集
*/
if(StartTask(FTMT_TASK_RUN_MODE_T1) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.10.2 StopTask

描述：

停止设备的数据采集工作。

声明：

```
int StopTask(int rMode);
```

参数：

rMode：输入参数，设备的当下工作模式。可接受值为：1，2，3，4，5。各值具体意义请参见 4.3 节。

返回值：

停止是否成功。可能值及其意义：

0: 成功，用宏 FTMT_ERROR_NONE 表示。

其他: 参见 4.2 节。

示例代码：

```
/*
* 停止设备T1的数据采集工作
*/
if(StopTask(FTMT_TASK_RUN_MODE_T1) != FTMT_ERROR_NONE)
{
    //错误处理代码
}
```

5.10.3 IsTaskCompleted

描述：

数据采集工作是否已经结束。

声明：

```
bool IsTaskCompleted();
```

参数：

无。

返回值：

设备是否已停止工作。可能值及其意义：

true: 已停止。

false: 未停止。

示例代码：

```
/*
* 停止设备T1的数据采集工作
*/
if(IsTaskCompleted ())
{
    //数据采集已停止
}
```

6 使用指南

6.1 驱动程序安装

直接安装 Cypress FX3 开发包即可。

6.2 DLL 加载

直接通过 Windows 系统函数 LoadLibrary 进行加载。示例代码：

```
_hDll = LoadLibrary(TEXT("smncsftmt.dll"));
if (_hDll == NULL)
{
      //错误处理
}
```

6.3 任务停止后处理

本 DLL 通过 API 接口与宿主程序进行交互，从而达到对 TCSPC 的控制。任务(T1, T2, T3 和数据符合)开始后，用户代码应调用接口 IsTaskCompleted 来检测任务是否已经完成，进而进行后续的数据处理过程。示例代码如下：

```
/*
* 启动设备进行T1数据采集
*/
if(StartTask(FTMT_TASK_RUN_MODE_T1) != FTMT_ERROR_NONE)
{
      //错误处理代码
}

//处理其他事物


/*
* 停止设备T1的数据采集工作
*/
if(StopTask(FTMT_TASK_RUN_MODE_T1) != FTMT_ERROR_NONE)
{
      //错误处理代码
}


While(!IsTaskCompleted(FTMT_TASK_RUN_MODE_T1))
{
      //处理其他事物
}

//对说采集的T1数据进行处理
```

对其他的任务应采取同样的处理方式。

7 参考代码详解

7.1 T1 数据采集

7.1.1 T1 数据采集配置流程

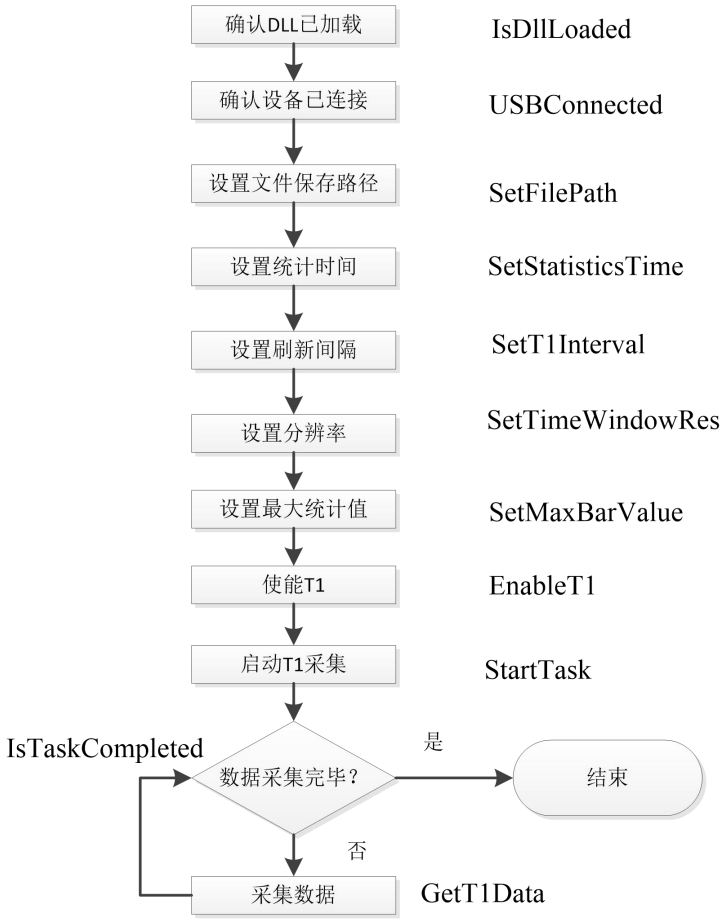

图 3　T1 数据采集配置流程

7.1.2 T1 数据采集代码示例

```
DllWrapper*    _dwrap = new DllWrapper(); // DllWrapper  是对DLL的封装，便于调用
if (_dwrap->IsDllLoaded()) //确认DLL是否加载成功
{
        System::Windows::Forms::MessageBox::Show("DLL is not imported Correctly!");
        return;
}

if (!_dwrap->USBConnected()) //确认USB已经与设备连接
{
        System::Windows::Forms::MessageBox::Show("USB is not Connected!");
        return;
}

if (_dwrap->SetFilePath(FTMT_TASK_RUN_MODE_T1, "E:\\Siminics\\data") != FTMT_ERROR_NONE) //设置数据文件保存位置
{
        System::Windows::Forms::MessageBox::Show("Call of SetPath Failed!");
        return;
```

```
        }

        if (_dwrap->SetStatisticsTime(FTMT_TASK_RUN_MODE_T1, 10.0f)!= FTMT_ERROR_NONE)//10s统计时间
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStatisticsTime Failed!");
            return;
        }

        if (_dwrap-> SetT1Interval (0.1f)!= FTMT_ERROR_NONE)//0.1s刷新时间
        {
            System::Windows::Forms::MessageBox::Show("Call of SetT1Interval Failed!");
            return;
        }

        if (_dwrap->SetFileMode(FTMT_TASK_RUN_MODE_T1, FTMT_FILE_SAVE_MODE_TEXT, FTMT_FILE_SERIES_MODE_LAST) !=
FTMT_ERROR_NONE)//数据保存模式：T1，保存为文件，只保存最后一次更新
        {
            System::Windows::Forms::MessageBox::Show("Call of SetFileMode Failed!");
            return;
        }


        if (_dwrap->EnableT1(FTMT_BOARD_A, CH_MASK_0)!= FTMT_ERROR_NONE)//使能BOARD A, CHANNEL 0
        {
            System::Windows::Forms::MessageBox::Show("Call of EnableT1 Failed!");
            return;
        }

        if (_dwrap->SetStartFreqDiv(FTMT_START_FREQ_DIV_1)!= FTMT_ERROR_NONE) //设置分频:1
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStartFreqDiv Failed!");
            return;
        }

        if (_dwrap->SetTimeWindowRes(FTMT_WIN_RES_PS_64)!= FTMT_ERROR_NONE)//设置分辨率：64ps
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStartFreqDiv Failed!");
            return;
        }

        if (_dwrap->SetMaxBarValue(100000ULL)!= FTMT_ERROR_NONE)//最大统计值
        {
            System::Windows::Forms::MessageBox::Show("Call of SetMaxBarValue Failed!");
            return;
        }

        if (_dwrap->StartTask(FTMT_TASK_RUN_MODE_T1)!= FTMT_ERROR_NONE)//启动 T1
        {
            System::Windows::Forms::MessageBox::Show("Call of StartTask Failed!");
            return;
        }

        unsigned int x_max;
        unsigned __int64 y_max;
        unsigned int r_cnt;
        unsigned __int64* d_ptr;
        try {
            d_ptr = new unsigned __int64[FTMT_T1_DATA_LEN]; //分配用于获取T1数据的内存
        }
        catch (std::bad_alloc)
        {
            System::Windows::Forms::MessageBox::Show("Allocate Memory Failed!");
            return;
        }


        while (!_dwrap->IsTaskCompleted()) //T1任务是否完成
        {
            Sleep(500);
            _dwrap->GetT1Status(&x_max, &y_max, &r_cnt);
```

```
                if (_dwrap->GetT1Data(d_ptr, 0, 0) != FTMT_ERROR_NONE)//brd:0, chl:0
                {
                        System::Windows::Forms::MessageBox::Show("GetT1Data Failed!");
                        return;
                }
                //Code To Display Data  数据显示采用LiveCharts
                ChartValues<ObservablePoint^>^ cv_ref = gcnew ChartValues<ObservablePoint^>();
                System::Collections::Generic::List<ObservablePoint^>^ l_ref = gcnew    System::Collections::Generic::List<ObservablePoint^>();
                l_ref->Add(gcnew ObservablePoint(5200, 0));

                for (int i = 0; i < FTMT_T1_DATA_LEN; i++)
                {
                        if(d_ptr[i]!=0)
                                l_ref->Add(gcnew ObservablePoint(i, d_ptr[i]));
                }
                l_ref->Add(gcnew ObservablePoint(5400, 0));
                cv_ref->AddRange(l_ref);


                ColumnSeries^ cs_ref = gcnew ColumnSeries();
                cs_ref->Title = "Test";
                cs_ref->Values = cv_ref;

                SeriesCollection^ sc_ref = gcnew SeriesCollection();
                sc_ref->Add(cs_ref);

                cartesianChart1->Series = sc_ref;
                return;

        }
```

使用此代码需要注意：

1. 动态加载 DLL。用户可以定义自己的 DLL Wrapper。
2. 波形显示使用了 LiveCharts 库，详情参见 https://lvcharts.net。

7.2 T2 数据采集

7.2.1 T2 数据采集配置流程

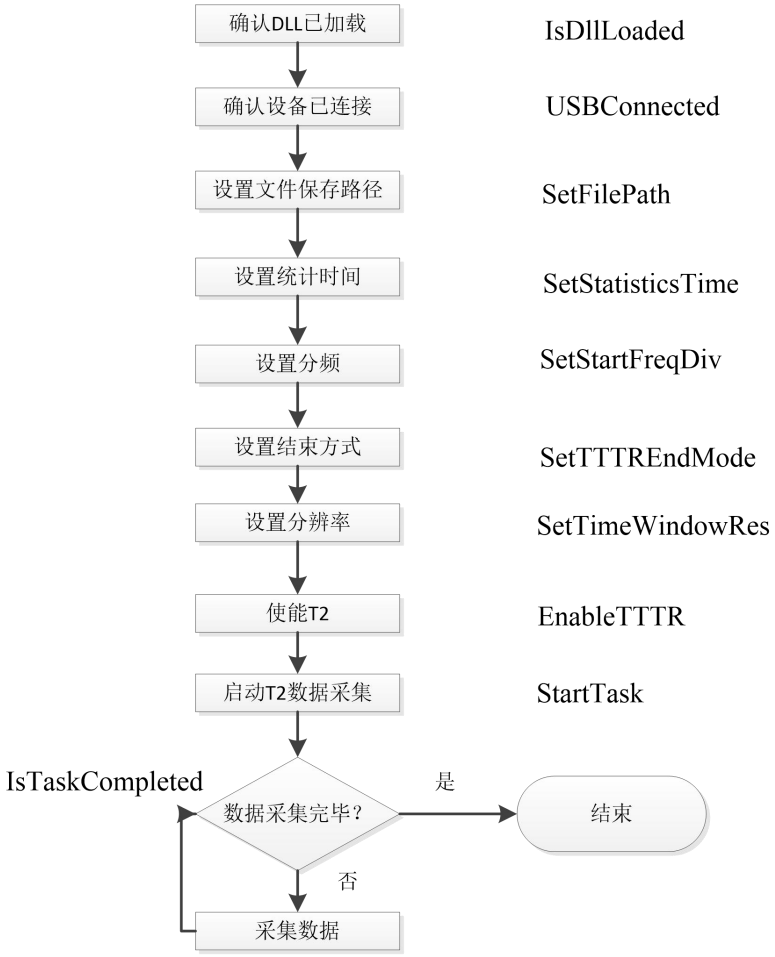

图 4 T2 数据采集配置流程

7.2.2 T2 数据采集代码示例

```
if (!this->_dwrap->IsDllLoaded())
{
	System::Windows::Forms::MessageBox::Show("DLL is not imported Correctly!");
	return;
}

if (!_dwrap->USBConnected())
{
	System::Windows::Forms::MessageBox::Show("USB is not Connected!");
	return;
}

if (_dwrap->SetPath(FTMT_TASK_RUN_MODE_T2, "E:\\Siminics\\data") != FTMT_ERROR_NONE)
{
	System::Windows::Forms::MessageBox::Show("Call of SetPath Failed!");
	return;
}

if (_dwrap->SetStatisticsTime(FTMT_TASK_RUN_MODE_T2, 10.0f) != FTMT_ERROR_NONE)//统计时间10s
{
	System::Windows::Forms::MessageBox::Show("Call of SetStatisticsTime Failed!");
	return;
}

if (_dwrap->SetFileMode(FTMT_TASK_RUN_MODE_T2, FTMT_FILE_SAVE_MODE_BIN, FTMT_FILE_SERIES_MODE_EACH) !=
FTMT_ERROR_NONE)//T2模式，保存二进制，第三个参数忽略
{
	System::Windows::Forms::MessageBox::Show("Call of SetFileMode Failed!");
```

```
            return;
        }

        if (_dwrap->EnableTTTR(FTMT_BOARD_A, CH_MASK_0) != FTMT_ERROR_NONE)//BOARD A, CHANNEL 0
        {
            System::Windows::Forms::MessageBox::Show("Call of EnableTTTR Failed!");
            return;
        }

        if (_dwrap->SetStartFreqDiv(FTMT_START_FREQ_DIV_1) != FTMT_ERROR_NONE) //Frequency Division:1
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStartFreqDiv Failed!");
            return;
        }

        if (_dwrap->SetTTTREndMode (FTMT_TTTR_END_MODE_TIME， 0) != FTMT_ERROR_NONE) //时间结束
        {
            System::Windows::Forms::MessageBox::Show("Call of SetTTTREndMode Failed!");
            return;
        }


        if (_dwrap->StartTask(FTMT_TASK_RUN_MODE_T2) != FTMT_ERROR_NONE)//StartTask T2
        {
            System::Windows::Forms::MessageBox::Show("Call of StartTask Failed!");
            return;
        }

        while (!this->_dwrap->IsTaskCompleted())
        {
            Sleep(100);
        }
        System::Windows::Forms::MessageBox::Show("T2 Completed!");
```

7.3 T3 数据采集

7.3.1 T3 数据采集配置流程

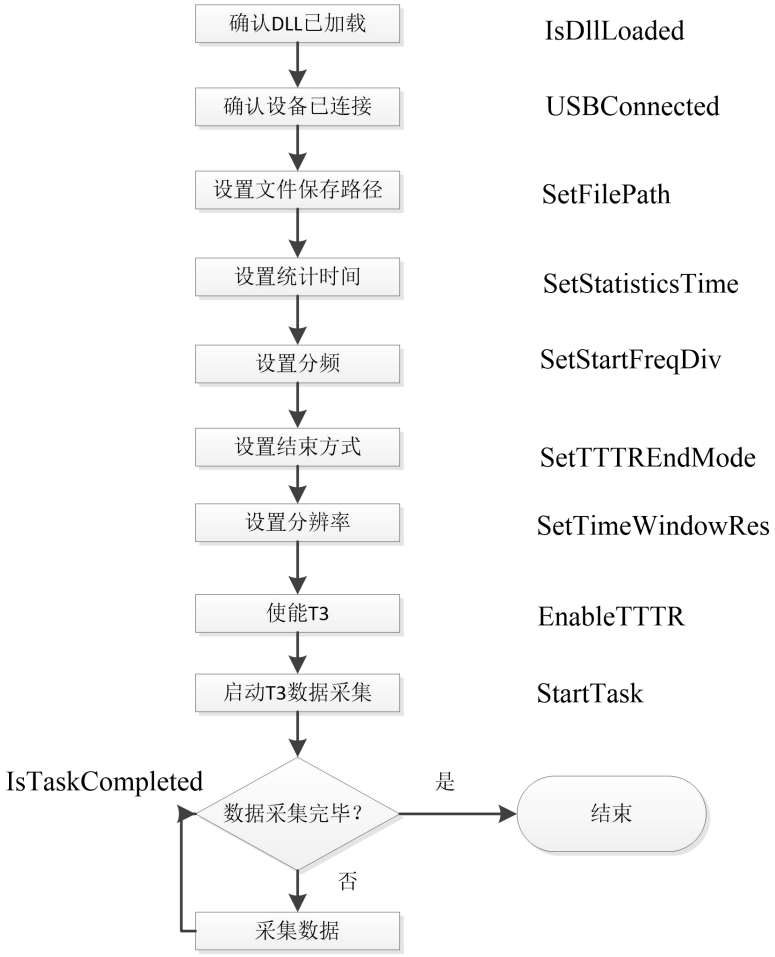

图 5　T3 数据采集配置流程

7.3.2 T3 数据采集代码示例

T3 数据采集代码和 T2 相似，也抄录如下：

```
if (!this->_dwrap->IsDllLoaded())
 {
        System::Windows::Forms::MessageBox::Show("DLL is not imported Correctly!");
        return;
 }

 if (!_dwrap->USBConnected())
 {
        System::Windows::Forms::MessageBox::Show("USB is not Connected!");
        return;
 }

 if (_dwrap->SetPath(FTMT_TASK_RUN_MODE_T3,  "E:\\Siminics\\data") != FTMT_ERROR_NONE)
 {
        System::Windows::Forms::MessageBox::Show("Call of SetPath Failed!");
        return;
 }

 if (_dwrap->SetStatisticsTime(FTMT_TASK_RUN_MODE_T3, 10.0f) != FTMT_ERROR_NONE)// 统计时间10s
 {
        System::Windows::Forms::MessageBox::Show("Call of SetStatisticsTime Failed!");
        return;
 }
```

```
            if (_dwrap->SetFileMode(FTMT_TASK_RUN_MODE_T3, FTMT_FILE_SAVE_MODE_BIN, FTMT_FILE_SERIES_MODE_EACH) !=
FTMT_ERROR_NONE)//
            {
                System::Windows::Forms::MessageBox::Show("Call of SetFileMode Failed!");
                return;
            }

            if (_dwrap->EnableTTTR(FTMT_BOARD_A, CH_MASK_0) != FTMT_ERROR_NONE)//BOARD A, CHANNEL 0
            {
                System::Windows::Forms::MessageBox::Show("Call of EnableTTTR Failed!");
                return;
            }

            if (_dwrap->SetStartFreqDiv(FTMT_START_FREQ_DIV_1) != FTMT_ERROR_NONE) //Frequency Division:1
            {
                System::Windows::Forms::MessageBox::Show("Call of SetStartFreqDiv Failed!");
                return;
            }

            if (_dwrap->SetTTTREndMode (FTMT_TTTR_END_MODE_EVENT， 1000) != FTMT_ERROR_NONE) //事件结束
            {
                System::Windows::Forms::MessageBox::Show("Call of SetTTTREndMode Failed!");
                return;
            }


            if (_dwrap->StartTask(FTMT_TASK_RUN_MODE_T3) != FTMT_ERROR_NONE)//StartTask T1
            {
                System::Windows::Forms::MessageBox::Show("Call of StartTask Failed!");
                return;
            }

            while (!this->_dwrap->IsTaskCompleted())
            {
                Sleep(100);
            }
            System::Windows::Forms::MessageBox::Show("T3 Completed!");
```

请注意，对于 T2/T3 模式，在进行长时间或高速数据采集时，数据保存方式请选择 FTMT_FILE_SAVE_MODE_BIN，此模式直接保存原始数据，速度最快。

7.4 数据符合

```
        if (!this->_dwrap->IsDllLoaded())
        {
            System::Windows::Forms::MessageBox::Show("DLL is not imported Correctly!");
            return;
        }

        if (!_dwrap->USBConnected())
        {
            System::Windows::Forms::MessageBox::Show("USB is not Connected!");
            return;
        }

        if (_dwrap->SetPath(FTMT_TASK_RUN_MODE_CO,"E:\\Siminics\\data") != FTMT_ERROR_NONE)
        {
            System::Windows::Forms::MessageBox::Show("Call of SetPath Failed!");
            return;
        }

        if (_dwrap->SetStatisticsTime(FTMT_TASK_RUN_MODE_CO, 10.0f) != FTMT_ERROR_NONE)
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStatisticsTime Failed!");
            return;
        }

        if (_dwrap->SetFileMode(FTMT_TASK_RUN_MODE_CO, FTMT_FILE_SAVE_MODE_BIN, FTMT_FILE_SERIES_MODE_EACH) !=
FTMT_ERROR_NONE)
```

```
        {
            System::Windows::Forms::MessageBox::Show("Call of SetFileMode Failed!");
            return;
        }

        if (_dwrap->EnableTTTR(FTMT_BOARD_A, CH_MASK_0) != FTMT_ERROR_NONE)//BOARD A, CHANNEL 0
        {
            System::Windows::Forms::MessageBox::Show("Call of EnableTTTR Failed!");
            return;
        }

        if (_dwrap->SetStartFreqDiv(FTMT_START_FREQ_DIV_1) != FTMT_ERROR_NONE) //Frequency Division:1
        {
            System::Windows::Forms::MessageBox::Show("Call of SetStartFreqDiv Failed!");
            return;
        }

        /*Code For Data Match*/
        if (_dwrap->dmtch_Enable(true) != FTMT_ERROR_NONE) //开启数据符合模式
        {
            System::Windows::Forms::MessageBox::Show("Call of dmtch_Enable Failed!");
            return;
        }

        if (_dwrap->dmtch_Initialize() != FTMT_ERROR_NONE) //数据符合模式初始化
        {
            System::Windows::Forms::MessageBox::Show("Call of dmtch_Initialize Failed!");
            return;
        }

        int strat1[] = { 0, 1, 2 };
        int strat2[] = { 0, 5, 7 };
        if (_dwrap->dmtch_AddStrategy(1, strat1, sizeof(strat1) / sizeof(int), true, 1000) != FTMT_ERROR_NONE) //添加数据符合策略1
        {
            System::Windows::Forms::MessageBox::Show("Call of dmtch_AddStrategy Failed for strat1!");
            return;
        }
        if (_dwrap->dmtch_AddStrategy(2, strat2, sizeof(strat2) / sizeof(int), true, 1000) != FTMT_ERROR_NONE) //添加数据符合策略2

        {
            System::Windows::Forms::MessageBox::Show("Call of dmtch_AddStrategy Failed for strat2!");
            return;
        }


        if (_dwrap->StartTask(FTMT_TASK_RUN_MODE_CO) != FTMT_ERROR_NONE)//StartTask T3
        {
            System::Windows::Forms::MessageBox::Show("Call of StartTask Failed!");
            return;
        }

        unsigned __int64 strat1_cnt;
        unsigned __int64 strat2_cnt;
        unsigned __int64 chl_cnt[FT1080_CHANNEL_NUM];

        while (!this->_dwrap->IsTaskCompleted())
        {
            strat1_cnt = _dwrap->dmtch_GetStratgyCount(1); //获取策略1的符合数
            strat2_cnt = _dwrap->dmtch_GetStratgyCount(2); //获取策略2的符合数
            if ((strat1_cnt < 0) || (strat2_cnt < 0))
            {
                System::Windows::Forms::MessageBox::Show("Call of dmtch_GetStratgyCount Failed!");
                return;
            }
            if (_dwrap->dmtch_GetChlCount(chl_cnt) != FTMT_ERROR_NONE) //获取每个通道的事件统计数
            {
                System::Windows::Forms::MessageBox::Show("Call of dmtch_GetChlCount Failed!");
                return;
            }
            /*
```

```
            *请在此处放置数据符合结果处理代码
            */
        }

        if (_dwrap->dmtch_Enable(false) != FTMT_ERROR_NONE) //关闭数据符合功能
        {
            System::Windows::Forms::MessageBox::Show("Call of dmtch_Enable Failed!");
            return;
        }

        System::Windows::Forms::MessageBox::Show("Data Match Completed!");
```