版本：2.4

六通道单光子计数器使用说明

目录

1 快速设置

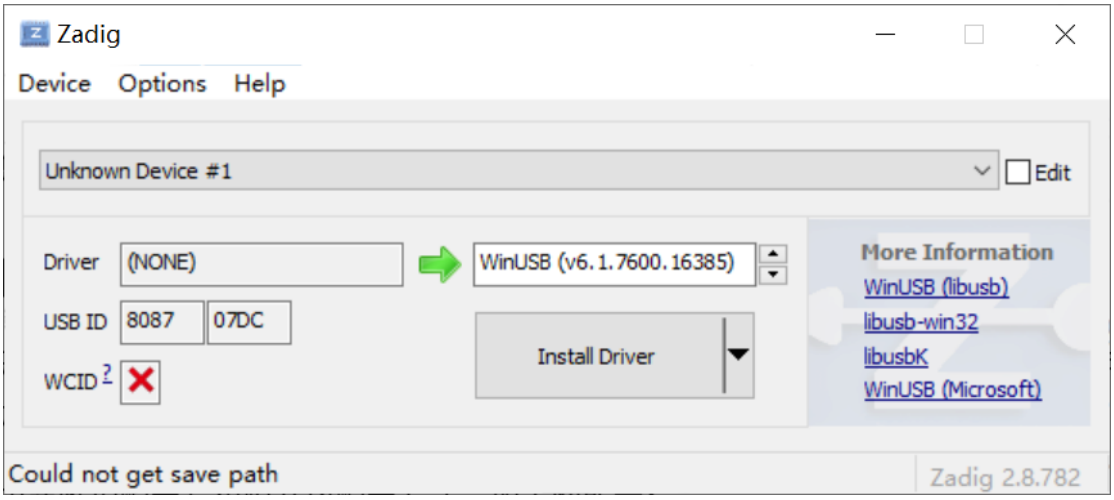
1.1 上位机配置

建议上位机使用 Windows 11 操作系统， 内存 16GB 及以上，并建议配备 SSD 固态硬盘。使用笔记本电脑的， 建议插电使用。

1.2 安装 USB 驱动

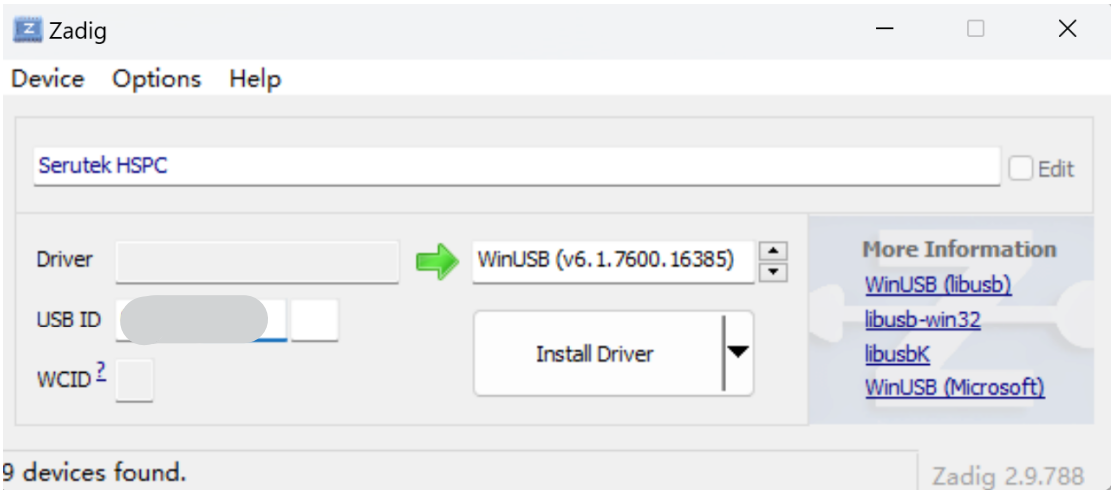
使用 Zadig-2.9 安装驱动。Zadig 已包含在安装压缩包中。

将计数器与上位机通过 USB3.0 线缆连接后，打开计数器电源。双击运行 Zadig。



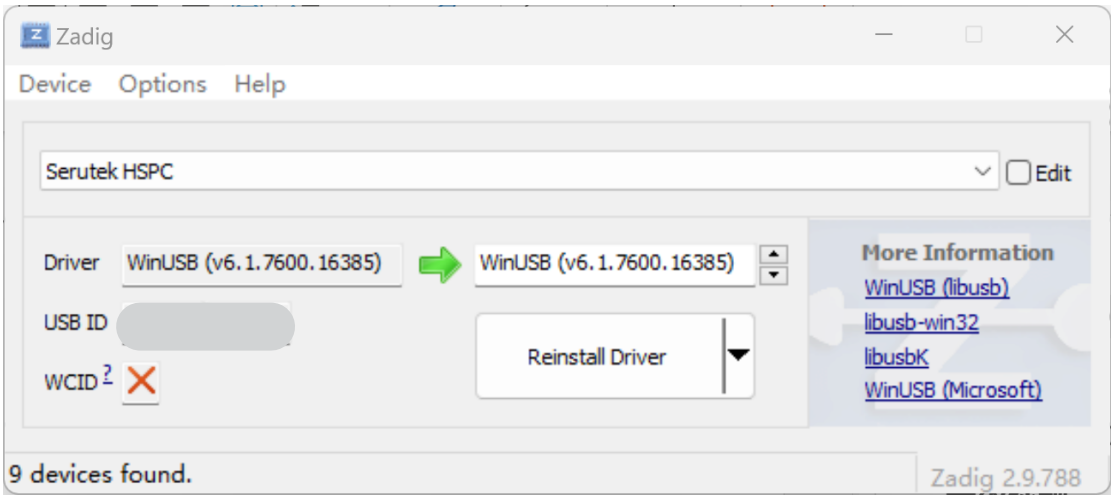
在菜单栏点击 Device->Create New Device

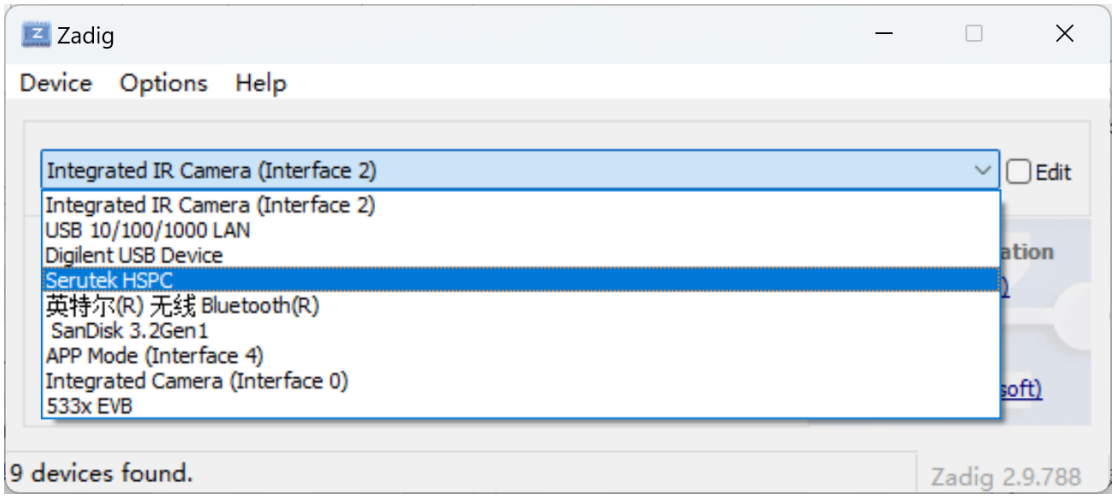
在 USB ID 栏填写如下图信息， 然后点击 Install Driver



等安装完成后， 关闭计数器，等待 2 秒后， 重新连接电源。如果驱动安装成功，上位机会发出提示音，在 zadig 软件中， 点击 Options->List All Devices，应当能够看到 Serutek HSPC 设备。

请确认 Serutek HSPC 设备驱动(Driver)版本如下图所示，如果不一致，请点击升级或重新安装驱动。





1.3 安装万兆网卡及配置

如果只使用 USB3.0，则无需安装万兆网卡，可跳过本章节。

上位机操作系统为 Win11 的，需要使用 X710 双光口万兆网卡。Win10 的可以使用 82599 或 X520 双光口万兆网卡，此款网卡不支持 Win11。

网卡配置： MTU9000

中断裁决：高

万兆网卡 IP 地址：

#1 192.168.1.4

#2 192.168.1.3

网卡配置如有问题请联系瑟如电子远程操作完成。

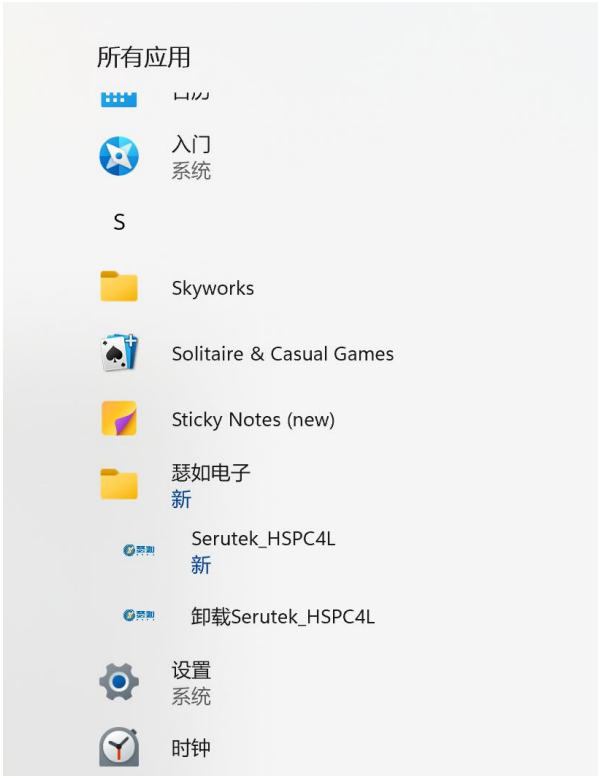
1.4 安装上位机程序

安装 DotNet 8.0 桌面运行时。可去微软官方下载，下载地址：

https://builds.dotnet.microsoft.com/dotnet/WindowsDesktop/8.0.15/windowsdesktop-runtime-8.0.15-win-x64.exe

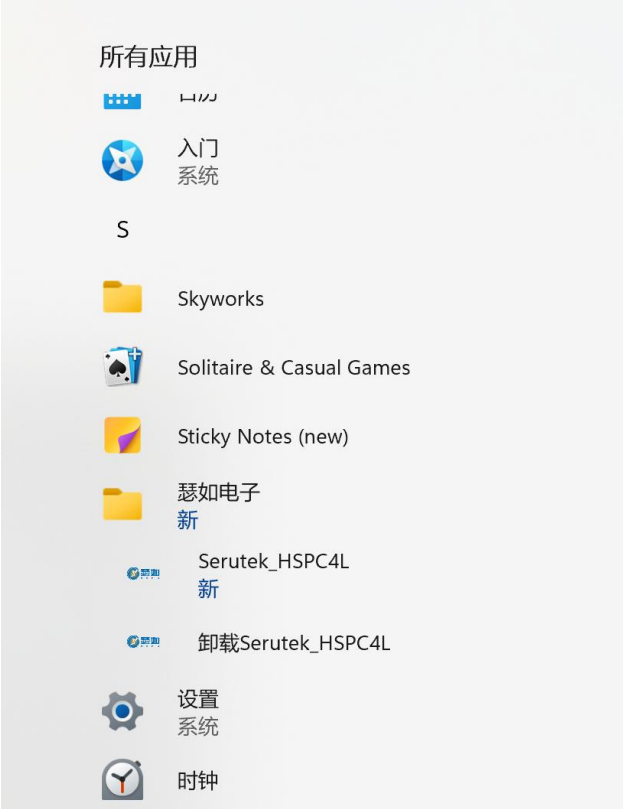
瑟如电子提供了安装程序包“Serutek_HSPC6.msi”。双击安装包按提示安装即可。安装结束后，在桌面可点击快捷方式“SeruTek HSPC4L”打开上位机软件：

或者点击程序菜单栏->瑟如电子->SeruTek HSPC4L，打开上位机软件。



1.5 卸载上位机软件

如需升级上位机软件， 那么需要先卸载老版本的软件。卸载方法如下：点击程序菜单栏->瑟如电子->卸载 SeruTek_HSPC4L 即可，如下图。



1.6 连接线缆

1. 开启计数器。计数器初始化完成后，LAN 口的黄色灯点亮。如果黄色 LED 灯闪烁，则表示计数器初始化失败，可能是固件未经授权或内部时钟没有锁定等内部异常。
2. 开启上位机。
3. 用 USB3.0 公对公线连接计数器和上位机。特别注意需要接到电脑的 USB3.0 接口上，如果接到 USB2.0 接口上，那么系统将不能正常工作。USB 连接正常后，LAN 的绿色灯点亮。

4. 启动上位机软件 SRPC (Se Ru Photon Counter) 。SRPC 启动后会检测计数器是否连接，如果软件能够找到设备，那么在软件界面底部的状态栏会显示“USB 已连接”。反之显示“！USB 未连接”，软件界面为不可操作状态。计数器在不进行测量时，USB 是可热插拔的。如果计数器在测量时断开 USB 连接，那么有可能导致计数器内部错误，需要重启计数器。

2 工作原理与功能

2.1 工作原理

上海瑟如单光子计数器基于 TDC（时间数字转换）工作，记录输入事件对应的时间戳。 每个输入通道上配置一路 TDC，该通道上输入的电压脉冲信号的上升沿将触发 TDC 的记录动作，将该上升沿对应的时间戳记录并输出。

各通道上的 TDC 独立运行，但在时间上保持同步。当计数器接收到开始测量的指令， TDC 内部逻辑开始运行计数， 此时刻是时间戳的原点（或零点）。TDC 计数一定时间后会因溢出产生回滚， 当前版本回滚周期是 52000 多秒[1]，约为 14 小时，当测量停止后，时间戳复位。大多数情况下，单次测量的时间不会有 14 小时这么长， 因此用户无需担心时间戳溢出的问题。如果确实需要进行单次长时间的测量， 那么也可以对溢出后的时间戳进行补偿，具体方式请联系瑟如电子。

上位机通过 USB3.0 与计数器进行通信。所有测量功能：时间戳、时间差、直方图、以及强度测量都可以通过 USB3.0 完成。可使用万兆光口作为时间戳（T2）和时间差（T3）的输出端口。当使用万兆网作为数据输出端口时，可以持续输出最大 200Msps 的时间戳数据流。

输出时间戳流是本计数器的基本功能，其它的功能如时差输出、直方图输出等功能都是基于时间戳处理后得到的。

[1] 只要测量不横跨计数器回滚前后时刻， 就无须对测量结果进行补偿。如果需要进行长时间的持续测量，并有可能横跨回滚周期，请联系瑟如电子，告知数据处理方法

2.2 功能概述

2.2.1 通道数量和功能

本计数器具备 6 个独立通道。 标准版中，通道 1-4 为一组，通道 5~6 为一组， 每组具备 100Msps 的输出能力。 实际测试中，通道 5 或 6 可用作高频同步信号以及事件标识，其余通道用于测量探测器输出的单光子脉冲。

L 版中通道 1~6 的数据汇聚在一起后通过 USB3.0 输出，饱和计数率为 39Msps。

在《参数设置》页面，可以对每个通道的触发电压阈值、通道时延进行补偿。也能够关闭或开启某个指定的通道。同时可以对通道 1 设置抑释功能。

2.2.2 通道时延补偿

可独立地对单独某一个通道进行时延调整，调整范围约为 Int32 整型范围，步进为 1 ps。需要注意的是，该调整只是在时间戳上增加了一个补偿量，没有对

信号到达的时刻进行前移后滞后调整。因此，在输出时间差模式和实时直方图模式下，建议调整量控制小于开门信号的周期。例如：开门信号的周期为 10MHz，对应的信号周期为 100 ns，那么通道时延调整量应小于 100ns， 以保证内部计算时间差模块的正确运行。

2.2.3 通道触发电压设置

每个通道能够接受最大电压为 5V 的电信号，输入阻抗为 50 欧姆。每个通道配备比较器，比较器的阈值电压独立可调，调节范围为 0mV~4096mV，步进为 1mV。当输入信号电压大于设置的阈值电压时，比较器输出从低电平转换到高电平，形成上升沿，从而被 TDC 记录下来。

2.2.4 通道使能

可以独立地关闭/打开指定的通道。需要注意的是这里的使能不影响计数率的显示，但是会对和时间戳/时间差测量有关的功能有效。关闭了指定的通道后，该通道的测量数据不会出现在记录的文件中或显示在直方图中。

2.2.5 通道 1 的抑释功能

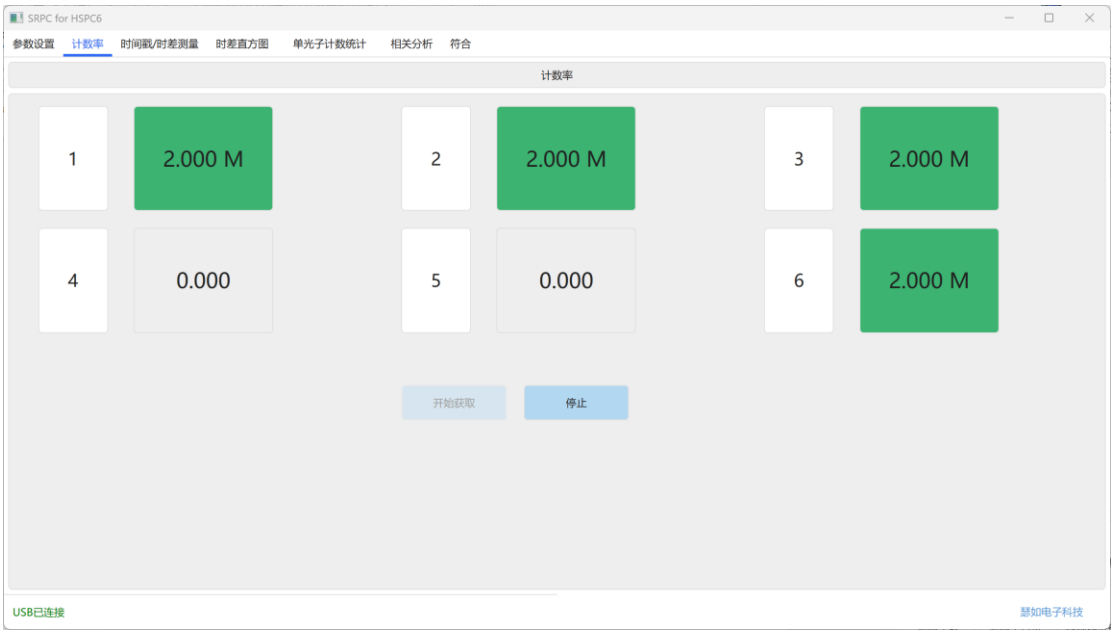
可以选择打开或关闭通道 1 的触发抑释功能。当打开抑释功能后，还可以调节抑释时间。抑释时间设置值为 0~127，单位约为 20 ns，最大可设置抑释时间约为 2.5 us。在抑释期间，触发功能关闭，即使有满足触发条件的脉冲到达，通道 1 也不会触发，相当于增大了通道 1 的死时间。

触发抑释功能可以在以下两个场景中使用：

1. 当同步通道接入 1 通道，而且计数率较高时。可以通过设置较大的抑释时间，降低同步信号实际的计数率，从而节省数据带宽。
2. 当同步信号下降沿较为缓慢，或信号质量较差，混入较多噪声时，使用触发抑释功能，可以让抑释时间覆盖整个下降沿，从而避免误触发。

2.2.6 计数率显示

该模式显示每个通道每秒钟记录的事件数量。一般在实验调试时用于测量探测到的光子强度。



2.2.7 时间戳输出模式（T2 模式）

在本模式下，单光子计数器将测量得到的时间戳原始数据通过 USB3.0 或 2 个万兆 SFP+光口（标准版）实时地发送到上位计算机。上位机启动同步接收功能，将接收到的时间戳以二进制文件保存。当使用万兆网口传输数据时，每次测

量将保存两个二进制文件，一个文件记录通道 1~4， 另一个记录通道 5~6 的数据。

二进制文件格式为：每 8 个字节对应一个时间戳。 因此，一个时间戳有 64 位，定义如下：

[63:57]	[56:0]
无符号整型，通道号	时间戳，有符号整型，LSB 单位： ps

具体解析方法，可参考瑟如电子提供的 Matlab 脚本。

2.2.8 时间差输出模式（T3 模式）

在 T3 模式下， 计数器输出各通道相对于 sync 同步信号的时间差，并同时记录对应的 sync 信号到达的时间戳，形成通常所说的 time-tagged time-resolved stream (TTTR 信息流)输出到上位机，保存为二进制文件。测量时，需要用户选择 sync 信号的通道作为 start 信号（开门）， 其余通道为 stop 信号（关门）。输出的是关门信号相对于开门信号滞后的时间。

在该模式下，用户需要选择开门信号的通道（Start）信号。测量开始后，计数器内部会根据时间戳计算其余通道相对于 sync 通道的时间差，并同时记录 sync 的时间戳。如果一个 sync 信号周期内没有光子到达， 那么不会记录改 sync 信号的时间戳。

T3 模式可以通过 USB3.0 也可以通过万兆网口输出。 当使用 USB3.0 时， 计数率最大为 39Msps， 当使用万兆网口时，使用 2 号万兆网口，计数率最大为 100Msps (在使用万兆网口时， 上位机会为两个网口分别创建一个二进制文件：xxx1.bin, xxx2.bin。 T3 模式下， TTTR 信息将被保存在第二个文件)。

TTTR 的二进制文件每条记录为 64bit。高 7 位表示通道号，其余位数根据是否为 sync 信号而有不同的意义。如果该记录是 sync 信号，那么低 57 位表示 sync 信号的时间戳。如果不是 sync 信号，那么[56:0]表示该通道相对于 sync 信号的时间差。

当通道为 sync 信号时：

[63:57]	[56:0]
无符号整型，通道号	时间戳，有符号整型，单位： ps

当通道不是 sync 信号时：

解析时间差文件时，每 8 个字节表示一个时间差记录，格式如下：

[63:57]	[56:0]
无符号整型，通道号	相对 sync 的时间差，ps

从时间差的数据格式可以看出，可以表示的时间差范围为±2^56 ps， 范围约为 40 小时，足以满足绝大部分应用。

以下数据截取自 TTTR 解析后得到的文本文件：

6: 197969

2: 2215

5: 2790

1: 2017

3: 2406

4: 2605

6: 364643

2: 2185

5: 2791

1: 2000

3: 2394

4: 2586

在该测量中， 通道 6 作为同步信号， 1，2，3,4,5 为光子探测通道。通道 6 的记录表示 sync 的时间戳，其余通道的记录表示光子相对于该 sync 的时间差。

2.2.9 实时直方图

实时直方图模式利用计数器输出的 T2 或 T3 数据实现高速时差直方图统计。用户需要设定一个通道为开门信号，并选择参数直方图绘制的关门信号（多选）。测量开始后，可同时显示多个通道相对于同步通道时间差的实时直方图。

用户可以在界面上选择直方图 bin 的大小和数量。 当同步信号频率较高（>20MHz）时，建议使用通道 5 或通道 6 作为同步信号。

在直方图界面，用户需要设置以下参数：

1. 统计模式，时间戳（T2）或时间差（T3）。该模式决定了计数器输出的是 T2 数据还是 T3 数据。当选择 T2 数据时，上位机软件根据计数器输出的时间戳计算出各通道相对于同步通道的时间差。而选择 T3 数据时，计数器直接输出内部计算得到的时间差数据，上位机软件根据时间差数据进行直方图统计。

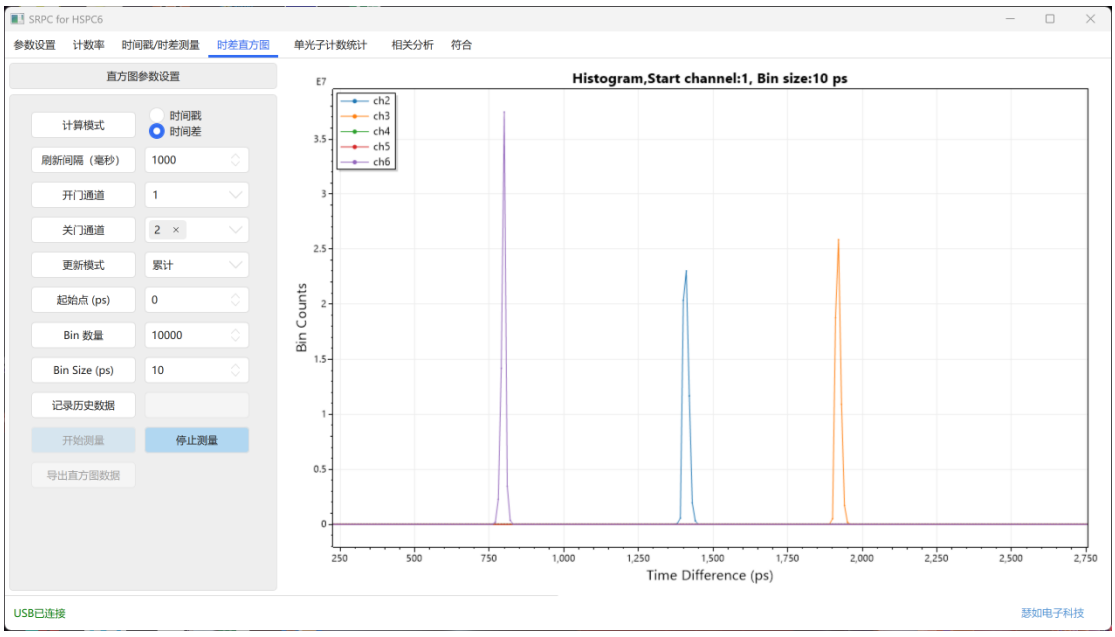
   在大多数应用中，T2 和 T3 模式得到的结果是一样的。但是当每一个同步脉冲间隔中，会有大量光子到达的情况下（比如数千或上万个光子），

计数器内部的 T3 模块无法处理正确处理时（HSPCL 模块能处理约 300 个光子/同步信号周期，HSPC 标准板约 5000 个光子/同步信号周期)，可使用时间戳模式（T2)。

2. 刷新时间间隔，单位毫秒。最小值为 500（毫秒)。该参数决定了间隔多久计算一次直方图。当计数率较高时（>20M)，建议选择小于等于 1000 毫秒的刷新间隔。
3. Scaling，放大率， 取值范围 0~40。当前版本使用 64bit 表示时间差，量程足够大。因此无需调整 scaling, 设为 0 即可。
4. 开门通道，可选择时差计算的开门通道，也就是时差计算的 start 信号。
5. 关门通道，也就是时差计算的 stop 信号。可同时选择多个关门通道，改变关门通道需要重新测量。
6. 更新模式。有两种模式，累积和刷新。 在累积模式下， 每次读取的计数值都于上次读取的该 bin 的计数值进行累加。而在刷新模式下， 显示的是当前读取的计数值，工作方式例如示波器，有些设备上也成为示波器模式。
7. 直方图起始点，单位 ps，默认值 0。当直方图的尖峰值远离同步信号 (时间差较大)，可以设置直方图的起始点或偏移量，减少对无用区域的统计计算。
8. Bin 数量，bin 的数量和 bin size(LSB)和 scale 这三个参数一起， 决定了直方图的量程。直方图量程= 2^scale * bin size * bin number (ps)。例如：Scale = 0; bin size = 10; bin number = 10000 时， 直方图量程为 2^0 * 10*10000 = 100000 ps = 100 ns。

尽管理论上 bin 的数量不受限制，但是 bin 数量越大，对上位机计算和显示的负荷就越大。建议在时间戳模式下，bin 的数量≤1e6；时间差模式下，bin 的数量小于等于 1e7。

9. Bin size(LSB)， 实际直方图每个 bin 的宽度= 2^scale * bin size(LSB)。
10. 记录历史数据，点击该按钮后可以选择保存路径。 当测量开始后， 会将直方图的数据记录以二进制数据的方式记录在文件中。 提供 Matlab 脚本来解析该二进制文件。
11. 导出直方图数据。当测量结束后，点击该按钮， 可以将最后一个刷新周期的直方图数据保存在用户指定的文件中。 保存格式为文本文件。 使用该功能时，建议将更新模式选择为累计。



2.2.10 光子计数统计

光子统计计数指统计一段时间内，某个通道上的光子数量。测量时可以选择两种模式。

一、 时间窗口模式。此模式下，用户需要指定统计窗口长度，单位是 us。计数器测量指定时间间隔内每个通道上的光子计数。

二、 同步通道触发模式。此模式下，用户需要定制同步信号的通道号。计数器测量每个通道上在两个同步信号之间到达的光子数。

测量开始后，上位机软件以每秒一次的频率读取计数器输出的计数值，并实时地绘制曲线。用户也可以选择将计数结果保存在二进制文件中，便于测量后分析。

光子强度二进制文件格式如下。每一条记录为 64 位（8 字节）。第一条记录（8 字节）是文件头，记录了同步通道、统计窗口长度等测量信息。接下去每 64 个字节为一组数据，其中每 8 个字节表示一个通道的计数值，从通道 1 依次到通道 6。当以时间窗口模式测量时，这一组数据代表一个窗口内 6 个通道的计数。

当以同步信号模式测量时，这一组数据代表一个同步脉冲与下一个同步脉冲之间，6 个通道的计数。

每一条记录的格式如下：

[63:40]	[39:32]	[31:0]
Unsigned int 计数值对应的同步脉冲的序号	Unsigned int 通道号	Unsigned int 计数值

在时间窗口模式下， 记录的**[63:40]**位，没有意义，可忽略。[39:32]8 位，表示通道号，记录的低 32 位表示计数值。例如：0x00000302000002A0 表示通道 2 的计数值为 672（0x2A0）,该计数值表示的是同步通道第 3 个脉冲到底 4 个脉冲之间，2 通道上的光子计数。

在同步信号模式下，记录的**[63:40]**位表示计数值对应的同步脉冲的序号。

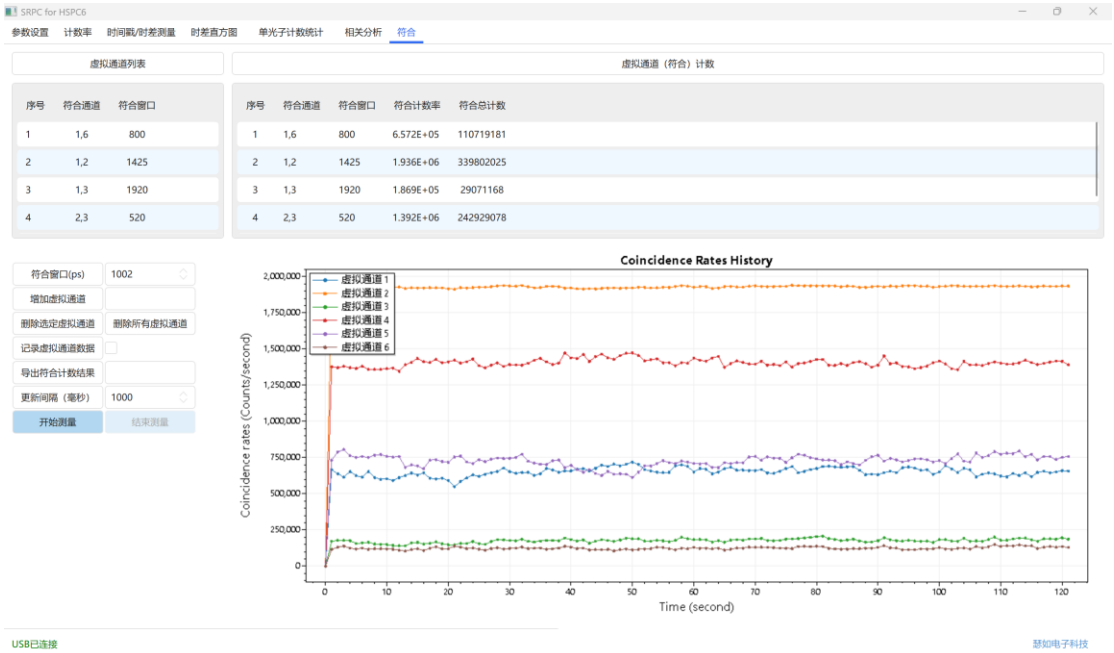
提供了 matlab 脚本文件，用来解析记录的光子强度二进制文件，将解析结果储存在 workspace 中。

报头格式如下：

[63:40]	[39:32]	[31:0]
同步信号的通道号 (sync)	0x00	时间窗口长度（us） (intense_window)

当 sync==0 时，表示测量以时间窗口模式进行，此时包头的第 4 字节表示时间窗口的长度，单位 us。当 sync == 1~6 时，表示测量以同步脉冲模式进行，同步信号通道为 sync 的值，此时，低 32 位的值可以忽略。

2.2.11 符合计数



本机提供强大灵活的符合计数功能。 用户可以设置多个符合策略，每个策略都可单独设置参与符合计算的通道和符合窗口的大小。符合策略的数量没有硬

性限制， 只受限于上位机的计算能力。 如果配备核心数量较多的 CPU， 可轻易实现同时进行上百个策略的符合计数。

本计数器中， 一条策略对应一个虚拟通道。 一个虚拟通道需要指定参与符合计算的通道和符合的串口宽度（单位 ps)， 通道之间以“,”间隔。输入参数后， 点击“增加虚拟通道”， 即可将该符合策略加入符合计算中。 当所有参与符合计数的通道上在指定窗口区间内都出现光子，才会认为“符合”从而触发计数，也就是说通道之间是“与”的关系。

开始测量后， 每一条符合策略的测量结果都会在右边栏中以表格的形式呈现。主要指标有符合计数率， 表示在单位时间内（1 秒）满足符合策略的计数；以及符合计数总数， 表示从测量开始到当前时刻，所有满足该符合策略的计数值。下方的图标绘制所有符合策略从测量开始到当前时刻的符合计数率的历史曲线。

用户可以增加、删除虚拟通道 (符合策略)。参与符合计数的通道之间以“,”相隔（英文逗号)。当需要对已经添加的虚拟通道进行修改时，可以双击需要修改的单元格，修改后摁回车键即可。

2.2.12 选择参考时钟

参考时钟是计数器的时频基准，在较长时间尺度上（大于 100us），参考时钟的稳定性对测量结果的影响开始显现。计数器支持两种参考时钟的设置

1. 内部 25MHz 晶振
2. 外部 10MHz 恒温晶振

25MHz 晶振上电开机后可以立即投入使用，是计数器的默认参考时钟。 如果需要使用外部时钟， 请确保接入时钟接口的信号为 10M 方波，建议上升沿小于 2ns，幅度小于 3.3V，抖动小于 10ps。 信号质量或时钟相位噪声较差的时钟会劣化计数器的测量性能。

3 测量流程

3.1 线缆连接

5. 开启上位机。
6. 用USB3.0公对公线连接计数器和上位机。计数器的USB3.0接口在前面板上，特别注意需要接到电脑的 USB3.0 接口上，如果接到 USB2.0 接口上，那么系统将不能正常工作。
7. 连接万兆网卡线缆。配备的万兆网卡线缆为 SFP+ DAC 万兆线。 计数器前面板上有两个万兆端口。 计数器右侧万兆端口必须连接到上位机万兆网卡靠近主板侧的端口， 计数器左侧端口必须连接到上位机万兆网卡远离主板侧的光口。
8. 开启计数器。
9. 启动测量控制软件

3.2 启动测量界面程序

1. 启动计数器上位机程序，双击桌面快捷方式：Serutek_HSPC4L

当需要重启计数器时，请先关闭计数器并关闭上位机程序。等计数器上电结束后再打开上位机程序。

3.3 设置触发阈值电压

1. 选择选项页“参数设置”。
2. 更改相应通道的阈值电压，单位毫伏，数值为正整数，最大 4096。
3. 点击“写入 DAC 值”按钮。
4. 触发阈值电压断电后不会保存。当计数器重新启动后，默认每个通道的阈值电压默认设为 512mV。

3.4 通道时延调整

1. 选择选项页“参数设置。
2. 更改相应通道的时延补偿值，数值可正可负。
3. 点击“更新通道时延 “。
4. 通道时延调整量断电后不会保存。当计数器重启后， 每个通道的默认时延调整量设为 0（ps)。

3.5 显示计数率

1. 点击选项页“计数率 “
2. 点击“开始获取“按钮，开始显示各通道计数率。
3. 点击“停止“停止计数率测量。
4. 请在开始其它测量项目前停止计数率测量。
5. 当进行计数率测量时，如果点击其它选项卡，程序会自动终止计数率测量。

3.6 时间戳/时间差测量

1. 点击选项页“时间戳/时间差测量“
2. 选择输出端口：USB3.0
3. 点击按钮“选择保存路径“，选择保存测量数据文件的路径。
4. 选择时间戳模式或时间差输出。

5. 开门通道。时间差模式下的开门通道。 在时间戳模式中忽略。
6. Scale 放大率，仅在时间差模式下有效。由于采用 64bit 表示时间差， 量程足够大， 因此 scale 设为 0 即可。
7. 定时测量， 在“测量 N 秒“右侧的数字框中键入希望测量的时差（单位：秒)。然后点击按钮”测量 N 秒“。测量过程中该按钮显示灰色，按钮恢复表示测量结束。
8. 手动控制测量。点“开始测量”，并在结束的时候，点击“结束测量“按钮。
9. 解析测量文件。 可以用提供的 matlab 脚本对保存的二进制测量数据文件进行解析。使用 proc_ts_usb.m 对时间戳数据（T2 时间戳模式）进行解析。使用 proc_t3_usb.m 对时间差数据（T3 模式）进行解析。注意需要可能需要修改解析脚本中测量文件的文件名，以及同步通道号：

```
close all
clear
fname = 'tstd.bin';
ch_start = 6;
fileID = fopen(fname);
```

3.7 实时直方图

参见 2.2.9 实时直方图的描述

3.8 符合计数

参见 2.2.11 符合计数章节。

3.9 光子计数统计

单光子强度类似于计数率测量，统计一段时间内，某个通道上的光子数量。

测量时可以选择两种模式。

三、 时间窗口模式。此模式下，用户需要指定统计窗口长度，单位是 us。计数器测量指定时间间隔内每个通道上的光子计数。

四、 同步通道触发模式。此模式下，用户需要定制同步信号的通道号。计数器测量每个通道上在两个同步信号之间到达的光子数。

测量开始后，上位机软件以每秒一次的频率读取计数器输出的计数值，并实时地绘制曲线。用户也可以选择将计数结果保存在二进制文件中，便于测量后分析。

测量时首先选择统计模式，时间窗口还是同步信号。根据选择的模式，选择

同步信号的通道号或者输入时间窗口的长度。

如果需要将测量结果保存在文件中， 那么需要选择记录文件的路径和文件名。

设置完测量参数后，点击开始测量。在绘图通道控制区域可以使能某个通道的数据绘制。当某个通道被取消勾选后，它的记录不会在右图被绘制。但是这个通道的数据在记录文件中还是会被保存。用户可以在测量过程中实时更改绘图的通道使能。

4 DLL 开发示例

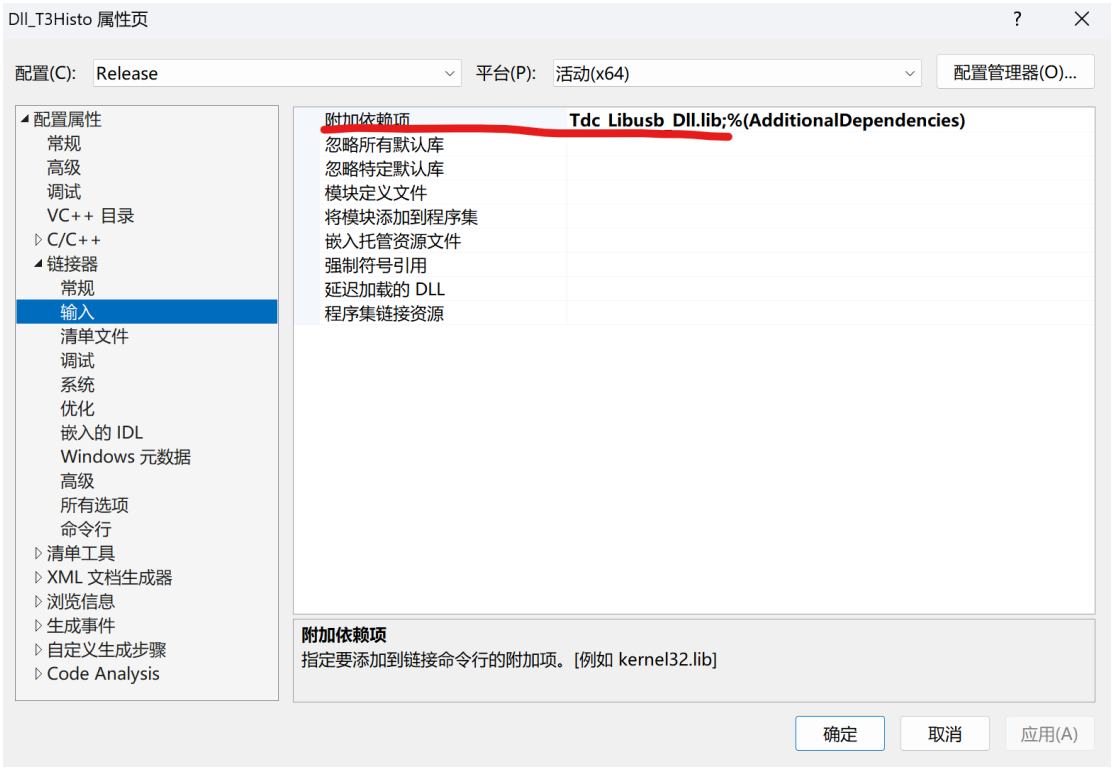
用户可使用提供的动态链接库和头文件在 Windows 下自主开发计数器控制程序，在线获取测量数据，实现自定义的自动测量流程以及数据处理应用。任何支持调用 DLL 的语言都可以支持。瑟如电子提供基 C++/Python/Labview 等语言的 DLL 示例程序

动态链接库和头文件可在上位机程序 Serutek_HSPC4L 的安装目录中找到。默认的安装目录为：C:\Program Files\Serutek\SeruTek_HSPC4_L。

需要注意的是：当调用 DLL 中的函数时，需要首先执行 LibUsb_Init()，初始化 LibUsb 环境。在退出程序前，必须执行 LibUsb_Exit()，退出 USB 环境。如果没有执行 LibUsb_Exit()，那么在下次执行 LibUsb_Init()时将发生异常。这种情况下只能将计数器重新上电。

4.1 新建 DLL C++应用程序

在 Visual Studio 中新建 C++工程， 并添加头文件 Tdc_Libusb_Dll.h。将 Tdc_Libusb_Dll.dll， Tdc_Libusb_Dll.lib 复制到工程文件夹中。并右击工程，进入属性设置页面，在连接器->输入->附加依赖项中添加 Tdc_Libusb_Dll.lib，如下图所示：



4.2 直方图统计示例

本示例使用 C++编写， 调用瑟如电子 SDK 动态链接库。先完成阈值电压设置、通道时延设置，之后设置直图统计的计算参数，并设置回调函数。示例中设置通道 1 为同步通道， 通道 2 和通道 5 参与直方图统计， 计数器测量通道 2 和通道 5 相对于通道 1 的时间差，并按照界面的设置（如 bin 的数量以及 bin 的宽度），以指定的时间间隔进行统计。在示例运行过程中，每隔指定的间隔，程序会读取上位机缓存中的 T3 原始数据进行直方图统计，并执行回调函数。用户可以在回调函数中读取如计数最大的 bin 的序号、当前光子计数等统计信息。

在进行直方图测量的同时，也可以将 T3 原始数据以二进制格式保存在文件中。保存文件有两种方式，一种是将完整的文件路径作为第一个参数传递给 HistoT3Measure 函数。另外，也可以如示例中调用 getHistoRawdata 函数获取 T3

数据，并将其写入文件。

以下将依次介绍示例程序中的代码：

首先是设定阈值电压：

dacwrite(i, 500);

i 取值范围是 0~5， 表示通道 1~6

第二步，设定通道时延，注意，这里第一个参数的取值范围是 1~6，表示通道 1~通道 6。 例如，以下语句对通道 1 的时间戳进行补偿， 在原来的时间戳上减去 2000 ps。

setuseroffset(1, -2000);

第三步，调用 HistoT3Measure，实现直方图测量。该函数参数较多，原型如下：

```
void HistoT3Measure(const char filename[], int mode, int chnum, int sync, int* hist_channels, int hist_chcount, int scale, int binsize, int binnum, int mt_ms, int period_ms, CountReady callback = nullptr);
```

const char filename[]:保存 T3 文件的目标路径，路径中避免含有中文字符。

int mode: 直方图模式，0 表示累计模式，1 表示刷新模式

int chnum： 表示计数器的最大通道数，可直接写为 10

int sync： 同步通道

int* hist_channels： 参与直方图统计的通道数组

int hist_chcount： 参与直方图统计的通道数量

int scale：时间差的缩放尺度， 没有必要缩放， 置为 0

int binsize： 直方图的 bin size，单位 ps

int Binnum: 直方图 bin 的数量

int mt_ms：测量持续的时间，单位毫秒

int period_ms：计算间隔，单位毫秒。每隔指定时间进行直方图统计。

CountReady callback： 回调函数，当每次计算间隔，直方图统计完成后调用该回调函数。

示例中调用参数设置如下：

```
HistoT3Measure("", 0, 10, 1, channels, 2, scale, binsize, binnum, 3100, 500, print_histmax);
```

保存文件路径为空字符串， 不保存文件；计数模式为：累计；最大通道数 10， 同步通道 1， 参与统计的通道为 int channels[2] = { 2,5 }，参与统计的通道数为 2， 缩放尺度为 0，测量时长为 3100 毫秒， 统计间隔为 500 毫秒，回调函数为 print_histmax。

在回调函数中，读取通道 2 和通道 5 直方图最大计数对应的 bin 序号、计数值，以及本轮读取数据的字节数，并打印。

int getHistoMax(int chan)，chan 取值 1~6， 返回该通道计数值最大的 bin 的序号。

UINT32 getHistoHitCountAtCh(int ch)， ch 取值 1~6，返回指定通道在本轮的计数值。

int getHistoProcBytes()， 返回本计算周期内处理的数据的字节数。

第四步，停止测量：

stopHistoT3Measure();

第五步：获取通道 2 直方图所有计数值并打印：

getBinCount(2, buffer, binnum);

其中 buffer 是指向一段缓存的指针，该缓存的大小至少为 binnum*4。

完整代码如下：

```
#include <iostream>
#include <vector>
#include <thread>
#include <iostream>
#include <fstream>
#include <sstream>
#include "Tdc_Libusb_Dll.h"


#define BUF_SIZE (1024*1024*128)
char destBuffer[BUF_SIZE];
volatile bool stop = false;

using namespace std;
UINT32 maxhistory[2][1024];
int byteshistory[1024];
int cntshistory[2][1024];
int index = 0;


//回调函数， 作为参数传递给HistoT3Measure， 当每一次直方图统计完成后， 都会调用该回调函数

void print_histmax()
{
    //获取通道2/5当前计数最大的bin 的序号(index)
    maxhistory[0][index] = getHistoMax(2);
    maxhistory[1][index] = getHistoMax(5);
    //得到原始数据的字节数
    byteshistory[index] = getHistoProcBytes();
    //得到通道2/5的光子计数
    cntshistory[0][index] = getHistoHitCountAtCh(2);
    cntshistory[1][index] = getHistoHitCountAtCh(5);
    printf("max is %d,  %d\r\n", maxhistory[0][index], maxhistory[1][index]);
    index++;
    int bytes_retrieved;
    //获取原始的T3数据，复制到destBuffer中，可以做一些用户自定义的处理。直方图测量函数本身可以保存T3原始数据到文件，
    //因此如果不是需要实时处理原始数据的话可以不用下面的函数。
    getHistoRawdata(destBuffer, &bytes_retrieved);
    printf("bytes retrieved is %d, \r\n", bytes_retrieved);
```

```
}



int main()
{
    std::cout << "Hello HistoT3 Test!\n";
    int ret = -1;

    //初始化USB环境
    LibUsb_Init();

    //设置阈值电压 500mV，dacwrite(0,xxx)设置通道1的阈值电压
    for (int i = 0; i < 6; i++)
    {
        ret = dacwrite(i, 500);
        if (ret != 0)
        {
            printf("write threshold voltage for channel %d failed\r\n", i+1);
            return -1;
        }
    }
    ret = 0;
    //设置通道时延
    ret += setuseroffset(1, -2000);
    ret += setuseroffset(2, 0);
    ret += setuseroffset(3, 0);
    ret += setuseroffset(4, 0);
    ret += setuseroffset(5, 0);
    ret += setuseroffset(6, 0);

    if (ret != 0)
    {
        printf("write channel offset failed\r\n");
        return -1;
    }
    //开始测量

    //定义参与直方图统计的通道， 这里是通道2和5
    int channels[2] = { 2,5 };
    //直方图bin 的个数
    int binnum = 1000;
    // bin size , 单位ps
    int binsize = 10;
```

```
    int scale = 0;
    index = 0;
    //启动直方图测量
    //第1个参数filename，保存T3原始数据的文件路径， 当为空时， 不保存T3原始数据
    //第2个参数mode, 0 为累计模式， 1为刷新模式
    //第3个参数chnum, 总的通道数， 可以用固定值10
    //第4个参数sync， 通道通道号， 为1时表示通道1是同步通道
    //第5个参数hist_channels,参与直方图统计的通道数组，
    //第6个参数hist_chcount， 参与直方图统计的通道个数，可以是参与直方图统计的通道数组的大小，也可以设置为比它小的值，比如数组长度是3， 但是这个参数可以设为1，2或3
    //第7个参数scale，缩放尺度，由于64bit 时间戳量程足够大， 所以缩放尺度可以固定为0
    //第8个参数 binsize
    //第9个参数 binnum， bin的数量， binsize * binnum 就是直方图统计的最大范围。可以根据需要修改。binnum越多， 计算负担越重
    //第10个参数 mt_ms, 测量持续的时间，单位毫秒（ms）
    //第11个参数 period_ms 直方图统计的时间间隔，单位毫秒(ms)， 指定每隔多久读取一次原始数据，并直方图统计。在高计数率下，需要考虑数据量不能大于内部缓存， 内部缓存为256MB。比如计数率为32M，那么每秒数据量为128MB，那么建议计算间隔小于2秒。推荐使用500ms~1000ms的计算间隔
    //第12个参数 回调函数， 可以不传递次参数， 意味着不需要回调函数。每次直方图统计后会调用回调函数。
    HistoT3Measure("", 0, 10, 1, channels, 2, scale, binsize, binnum, 3100, 500, print_histmax);
    std::this_thread::sleep_for(std::chrono::milliseconds(3200));
    //停止直方图测量
    stopHistoT3Measure();

    //准备缓存，用来存放直方图数据
    uint32_t* buffer = (uint32_t*)malloc(4 * binnum);
    //获取通道2的直方图统计数据
    getBinCount(2, buffer, binnum);

    //打印直方图数据
    for (int i = 0; i < binnum; i++)
    {
        printf("%d, %d %u\r\n", i, (i + 1) * binsize, buffer[i]);
    }

    //打印一些在回调函数中记录的历史信息
    for (int i = 0; i < index; i++)
    {
        printf("ch2: max %u, counts %d, bytes %d,\r\n", maxhistory[0][i], cntshistory[0][i], byteshistory[i]);
        printf("ch5: max %u, counts %d, bytes %d,\r\n", maxhistory[1][i],
```

```
cntshistory[1][i], byteshistory[i]);
    }

    printf("Done! Exit\r\n");

    /*getBinCount(3, buffer, binnum);
    for (int i = 0; i < 1000; i++)
    {
        printf("%d, %d %u\r\n", i, (i + 1) * binsize, buffer[i]);
    }*/

    printf("Done! Exit\r\n");
    //退出USB环境
    LibUsb_Exit();
    }
```

4.3 直方图 mapping 示例

示例演示了如何循环多次进行直方图测量，模拟需要 mapping 扫描的测试场景。每一次测量都会获取直方图统计信息，并将测量数据保存在一个独立的文件中。示例中循环测量了 20 次，原始数据保存在 20 个不同的二进制文件中。示例中，同步通道为 1 通道， 通道 2 和通道 5 参与直方图统计。通道 2 的统计数据在每一次测量完成后读取并打印。而通道 5 的测量数据在回调函数中读取并记录，在 20 次测量完成之后打印。以下是完整的源代码：

```
#include <iostream>
#include <vector>
#include <thread>
#include <iostream>
#include <fstream>
#include <sstream>
#include "Tdc_Libusb_Dll.h"


#define BUF_SIZE (1024*1024*128)
char destBuffer[BUF_SIZE];
//volatile bool stop = false;
```

```
using namespace std;
UINT32 maxhistory[1024];
int byteshistory[1024];
int cntshistory[1024];
int index = 0;

std::ofstream outF;
uint64_t totalbytes = 0;

//回调函数， 作为参数传递给HistoT3Measure， 当每一次直方图统计完成后， 都会调用该回调函数
void print_histmax();

int main()
{
    std::cout << "Hello HistoT3 Test!\n";
    int ret = -1;

    //初始化USB环境
    LibUsb_Init();

    //设置阈值电压 500mV，dacwrite(0,xxx)设置通道1的阈值电压
    for (int i = 0; i < 6; i++)
    {
        ret = dacwrite(i, 500);
        if (ret != 0)
        {
            printf("write threshold voltage for channel %d failed\r\n", i + 1);
            return -1;
        }
    }
    ret = 0;
    //设置通道时延
    ret += setuseroffset(1, -2000);
    ret += setuseroffset(2, 0);
    ret += setuseroffset(3, 0);
    ret += setuseroffset(4, 0);
    ret += setuseroffset(5, 0);
    ret += setuseroffset(6, 0);
    if (ret != 0)
    {
        printf("write channel offset failed\r\n");
        return -1;
    }
```

```
    //开始测量

    LibUsb_Exit();

    //定义参与直方图统计的通道， 这里是通道3和4
    int channels[2] = { 2,5};
    //直方图bin 的个数
    int binnum = 1000;
    // bin size , 单位ps
    int binsize = 10;
    int scale = 0;
    index = 0;
    //启动直方图测量
    //第1个参数filename，保存T3原始数据的文件路径， 当为空时， 不保存T3原始数据
    //第2个参数mode, 0 为累计模式， 1为刷新模式
    //第3个参数chnum, 总的通道数， 可以用固定值10
    //第4个参数sync， 通道通道号， 为1时表示通道1是同步通道
    //第5个参数hist_channels,参与直方图统计的通道数组，
    //第6个参数hist_chcount， 参与直方图统计的通道个数，可以是参与直方图统计的通道数组的大小，也可以设置为比它小的值，比如数组长度是3， 但是这个参数可以设为1，2或3
    //第7个参数scale，缩放尺度，由于64bit 时间戳量程足够大， 所以缩放尺度可以固定为0
    //第8个参数 binsize
    //第9个参数 binnum， bin的数量， binsize * binnum 就是直方图统计的最大范围。可以根据需要修改。binnum越多， 计算负担越重
    //第10个参数 mt_ms, 测量持续的时间，单位毫秒（ms）
    //第11个参数 period_ms 直方图统计的时间间隔，单位毫秒(ms)， 指定每隔多久读取一次原始数据，并直方图统计。在高计数率下，需要考虑数据量不能大于内部缓存， 内部缓存为256MB。比如计数率为32M，那么每秒数据量为128MB，那么建议计算间隔小于2秒。推荐使用500ms~1000ms的计算间隔
    //第12个参数 回调函数， 可以不传递次参数， 意味着不需要回调函数。每次直方图统计后会调用回调函数。
    //HistoT3Measure("", 0, 10, 1, channels, 1, scale, binsize, binnum, 1100, 500, print_histmax);

    //准备缓存，用来存放直方图数据
    //uint32_t* buffer = (uint32_t*)malloc(4 * binnum);



    totalbytes = 0;

    char filepath[80] = "D:\\temp\\t3loop\\";
```

```
    for (int i = 0; i < 20; i++)
    {

        char filename[20];
        char fullpath[80];
        strcpy(fullpath, filepath);
        sprintf(filename, "t3data_%d.bin", i + 1);
        strcat(fullpath, filename);

        //open file to write raw t3 data
        //outF.open(fullpath);

        printf("this is %d turn\r\n", i);
        //初始化USB环境
        LibUsb_Init();
        HistoT3Measure(fullpath, 0, 10, 1, channels, 2, scale, binsize, binnum, 100000,
500, print_histmax);
        std::this_thread::sleep_for(std::chrono::milliseconds(1100));
        //停止直方图测量
        stopHistoT3Measure();

        //获取5通道的直方图数据，复制到buffer中
        //getBinCount(5, buffer, binnum);

        //获取计数最高的bin 的index，并打印
        UINT32 max = getHistoMax(2);
        printf("ch2: max index is %u\r\n", max);



        //outF.flush();
        //outF.close();

        //退出USB环境
        LibUsb_Exit();
    }
    ////打印一些在回调函数中记录的历史信息
    for (int i = 0; i < index; i++)
    {
        printf("ch5: max %u, bytes %d, counts %d\r\n", maxhistory[i], byteshistory[i],
cntshistory[i]);
    }
    printf("Measurment Done! Exit!\r\n");
```

```
}

//回调函数， 作为参数传递给HistoT3Measure， 当每一次直方图统计完成后， 都会调用该回调函数
void print_histmax()
{
    //获取通道4当前计数最大的bin 的序号(index)
    maxhistory[index] = getHistoMax(5);
    //得到原始数据的字节数
    byteshistory[index] = getHistoProcBytes();
    //得到通道4的光子计数
    cntshistory[index] = getHistoHitCountAtCh(5);

    index++;
    int bytes_retrieved;
    //获取原始的T3数据，复制到destBuffer中，可以做一些用户自定义的处理。直方图测量函数本身可以保存T3原始数据到文件，
    //因此如果不是需要实时处理原始数据的话可以不用下面的函数。

    //getHistoRawdata((void*)destBuffer, &bytes_retrieved);
    //outF.write(destBuffer, bytes_retrieved);
}
```

4.4 符合计数示例

本示例中用 C++在 Visual Studio 2022 下编写。在本示例演示了如何设置通道触发阈值电压、通道时延，并创建了两个虚拟通道，也就是符合计数的策略。虚拟通道 1 是通道 1 和通道 2 的符合， 符合窗口为 1975ps。第 2 个虚拟通道是通道 1 和通道 5 的符合， 符合窗口是 1990ps。符合测量时长 3100 毫秒， 每隔 500 毫秒读取数据， 并统计符合计数、计算符合计数率。在回调函数中获取符合计数值、符合计数率、符合计算执行耗时等数据，并打印和保存数据，并在测量结束时打印历史数据。

启动符合计数测量的函数是：

void startCoinMeasure(int (*chan_list)[7], UINT64* windows_size, int list_row,

int mt_ms, int update_rate, CountReady callback = nullptr);

第 1 个参数 chan_list， 7 列二维数据，用来表示虚拟通道的构成，也就是符合策略。

第 2 个参数 windows_size， 窗口大小数组，为符合策略指定符合窗口的大小

第 3 个参数 list_row， 虚拟通道的个数

第 4 个参数 mt_ms， 测量持续的时间（毫秒）

第 5 个参数 update_rate， 符合计算的间隔时间（毫秒）

第 6 个参数 callback， 回调函数， 每次符合计算结束后会调用回调函数，可以不传递此参数。

在示例程序中， 一个参数是一个 2 行 7 列的二维数组，每一行代表一个虚拟通道，也就是符合策略。每一行的第一个元素表示有几个通道构成这个虚拟通道。在本示例中是 2，表示是 2 个通道之间的符合。每一行的后面 6 个元素表示参与符合计算的通道。对于第一个虚拟通道来说， 此行的第 2 和第 3 个元素分别为 1 和 2，表示通道 1 和通道 2 构成了此虚拟通道。

```
//设置 2 个虚拟通道， 第 1 个 1 和 2， 第 2 个 1 和 5
int chan_list[2][7];//第一个维度 2 表示有两个虚拟通道，如果虚拟通道有 3 个， 这里就是 3。7 是固定的
chan_list[0][0] = 2;//表示有两个通道参与符合
chan_list[0][1] = 1;
chan_list[0][2] = 2;

chan_list[1][0] = 2;//表示有两个通道参与符合
```

chan_list[1][1] = 1;

chan_list[1][2] = 5;

第 2 个参数是一个数组，表示对应的符合策略的符合窗口大小。 在示例中这个数组为：

UINT64 window_list[2] = { 1975,1990 };

表示第一符合策略（1,2 之间符合）的符合窗口是 1975ps, 第二个符合策略（1,5 之间）的符合窗口是 1990ps。此数组的长度应当等于符合策略的个数，也就是第一个参数的行数。

在示例中定义了回调函数 void print_coinRate()，并作为最后一个参数传递给 startCoinMeasure 函数。每次符合计算后， 都会调用该函数。在这个函数中调用了 getCoinCount(0)来获取第一个符合策略的计数值（getCoinCount(1)返回第二个符合策略的计数值)，getCoinRate(0)来获取第一个符合策略的计数率。获取的计数值保存在一个具有足够长度的数组中。当测量结束后，将整个测试过程中的符合计数值和计数率打印出来。

源代码如下：

```
#include <iostream>
#include "Tdc_Libusb_Dll.h"
#include <vector>
#include <thread>
using namespace std;

volatile uint64_t countHistory[10][1024];
volatile double rateHistory[10][1024];
volatile uint64_t etimeHistory[1024];
volatile uint32_t exetimeHistory[1024];
volatile int rbytesHistory[1024];
```

```
volatile int call_index;
ULONGLONG time_start, time_end;

void print_coinRate();

int main()
{
    std::cout << "Hello DLL Coincidence Test!\n";
    vector<uint64_t> CoinCnt;
    int ret = 0;


    //初始化USB环境s
    LibUsb_Init();

    printf("set trigger voltage\r\n");
    //设置阈值电压 500mV,注意，这个函数第1个参数为0时，表示通道1
    for (int i = 0; i < 6; i++)
    {
        ret = dacwrite(i, 500);
        if (ret != 0)
        {
            printf("set threshold voltage for channel %d failed\r\n", i + 1);
        }

    }
    printf("set channel delay\r\n");
    //设置通道时延

    ret += setuseroffset(1, -2000);
    ret += setuseroffset(2, 0);
    ret += setuseroffset(3, 0);
    ret += setuseroffset(4, 0);
    ret += setuseroffset(5, 0);
    ret += setuseroffset(6, 0);

    if (ret != 0)
    {
        printf("set channel offset failed!\r\n");
    }


    //设置2个虚拟通道， 第1个1和2， 第2个1和5
```

```
    int chan_list[2][7];//第一个维度2表示有两个虚拟通道，如果虚拟通道有3个， 这里就是
3。7是固定的
    chan_list[0][0] = 2;//表示有两个通道参与符合
    chan_list[0][1] = 1;
    chan_list[0][2] = 2;

    chan_list[1][0] = 2;//表示有两个通道参与符合
    chan_list[1][1] = 1;
    chan_list[1][2] = 5;

    //对两个虚拟通道分别使用不同的符合窗口
    UINT64 window_list[2] = { 1975,1990 };
    call_index = 0;
    printf("start measure\r\n");

    //开始符合测量
    //第1个参数chan_list， 虚拟通道数组
    //第2个参数windows_size， 窗口大小数组
    //第3个参数list_row， 虚拟通道的个数
    //第4个参数mt_ms， 测量持续的时间（毫秒）
    //第5个参数update_rate， 符合计算的间隔时间（毫秒）
    //第6个参数callback， 回调函数， 每次符合计算结束后会调用回调函数，可以不传递此参数
    startCoinMeasure(chan_list, window_list, 2, 3100, 500, print_coinRate);
    time_start = GetTickCount64();
    std::this_thread::sleep_for(std::chrono::milliseconds(3200));
    //停止符合测量
    stopCoinMeasure();

    for (int i = 0; i < call_index; i++)
    {
        printf("elapsed time: %llu , exe time %u, read bytes: %d\r\n", etimeHistory[i],
exetimeHistory[i], rbytesHistory[i]);
        printf("%d, coinRate0: %3.2e， coinRate1: %3.2e\r\n", i, rateHistory[0][i],
rateHistory[1][i]);
        printf("%d, coinCount0 %llu， coinCount %llu\r\n", i, countHistory[0][i],
countHistory[1][i]);
    }

    //退出USB环境
    LibUsb_Exit();
}

void print_coinRate()
{
```

```
    time_end = GetTickCount64();

    //读取第1个虚拟通道（符合策略）的计数值，需要每个周期都读取，读取后该通道的计数值会重置，否则计数值是累加的。
    countHistory[0][call_index] = getCoinCount(0);
    countHistory[1][call_index] = getCoinCount(1);
    //获取第1个虚拟通道（符合策略）的符合计数率
    double cnt0 = getCoinRate(0);
    double cnt1 = getCoinRate(1);
    rateHistory[0][call_index] = cnt0;
    rateHistory[1][call_index] = cnt1;
    //获取处理了多少字节数
    rbytesHistory[call_index] = getCoinProcBytes();
    //从开始测量到现在过了多少毫秒
    etimeHistory[call_index] = time_end - time_start;
    //符合运算的执行时间,单位ms
    exetimeHistory[call_index] = getCoinElapsedTime();
    call_index++;
    printf("time elapsed %llu\r\n", (time_end - time_start));
}
```

4.5 实时获取时间戳示例（python）

本示例在 Windows python 环境下使用 ctypes 库，导入动态链接库。完成触发阈值设置、通道时延补偿，并开始 T2 测量，每隔 0.2 秒，从上位机缓存中获取原始 T2 时间戳数据， 并将其写入指定的二进制文件中。完整源代码如下：

```
#coding=gb2312
#上海瑟如电子科技
#单光子计数器DLL控制及数据采集示例
# 在Windows下， 使用python调用动态链接库，实现时间戳测量，并实时获取测量数据，并写入指定文件
import ctypes;
from ctypes import *
import time

#从本地目录导入动态链接库
tDll = CDLL("./Tdc_Libusb_Dll.dll");

print('Hello Serutek!')

#申明一段缓存，大小32MBytes，可以储存4M个时间戳。高计数率下，或测量间隔较大，需要扩大缓
```

```
存大小
#使用ctypes的c_char类型
dBuffer = (c_char*32*1024*1024)()

#初始化USB环境
tDll.LibUsb_Init()

# write threshold voltage, start from 0 to 5
for ch in range(0,6):
    tDll.dacwrite(ch, 500)

# write channel offset, start from 1 to 6
for ch in range(1,7):
    tDll.setuseroffset(ch, 0)



totalbytes = 0
#写入当前目录下的tstd.bin
file = open('tstd.bin','wb')

#开始测量， 时间戳模式
tDll.start_tstd_mmf_usb(0)

#等待0.2秒
time.sleep(0.2)

#读取测量数据，参数1：最大读取数据字节数为缓存的大小，参数2：复制数据的目标地址
#返回值：实际读取的字节数
rbytes = tDll.get_tstd_mmf_usb_data(32*1024*1024, dBuffer)

#将缓存输入转换为byte array
bytes2w = ctypes.string_at(dBuffer, rbytes)
totalbytes += rbytes;

#写入文件
file.write(bytes2w)

#等待0.2秒
time.sleep(0.2)

rbytes = tDll.get_tstd_mmf_usb_data(32*1024*1024, dBuffer)
bytes2w = ctypes.string_at(dBuffer, rbytes)
file.write(bytes2w)
totalbytes += rbytes
```

```
#停止测量
tDll.stop_tstd_mmf_usb();

#读取残余数据写入文件， 也可以忽略
rbytes = tDll.get_tstd_mmf_usb_data(32*1024*1024, dBuffer)
bytes2w = ctypes.string_at(dBuffer, rbytes)
file.write(bytes2w)
totalbytes += rbytes
#关闭映射文件(必须)
tDll.close_tstd_mmf()

#关闭文件
file.flush()
file.close()

print("Measurement Done! " + str(totalbytes) + " bytes are written")

#退出USB环境
tDll.LibUsb_Exit();
```

4.6 光子强度监测示例（python）

本示例演示了如何在 python 中调用 DLL，统计各个通道在指定的时间窗口内的计数值以及如何以指定通道（示例中为通道 1）的信号作为 Marker 信号，来统计其余通道在 Marker 信号间隔内的计数。

用户调用 start_irrt_mmf 函数以启动强度监测，调用 stop_irrt_mmf()函数来停止强度监测。在测量中或测量停止后，都可以调用 get_irrt_mmf_usb_data 来获取所有通道的强度监测记录。示例中，获取的光子计数记录以二进制格式写入文件中。在退出程序前，需要调用 close_irrt_mmf，释放资源。

主要的函数 int start_irrt_mmf(int sync, int window_len) 有两个参数。第一个参数 sync 是同步信号所在的通道号，sync 等于 1~6 时，表示以同步信号模式测量，此时忽略第二个参数。如果 sync 等于 0，那么以时间窗口模式测量，时间窗

口长度由第二个参数 window_len 指定，单位 us。

示例中先调用 start_irrt_mmf 开启测量，然后循环 3 次，每次等待 0.4 秒后，读取计数强度，3 次读取结束后 stop_irrt_mmf。也可以将 start 或 stop 放在循环里。在退出程序前记得调用 close_irrt_mmf()释放资源。

以下是 python 源代码：

```
#coding=gb2312
#上海瑟如电子科技
#单光子计数器DLL控制及数据采集示例
# 在Windows下， 使用python调用动态链接库，实现时间戳测量，并实时获取测量数据，并写入指定文件
import ctypes;
from ctypes import *
import time

#从本地目录导入动态链接库
tDll = CDLL("./Tdc_Libusb_Dll.dll");

print('Hello Serutek!')

#申明一段缓存，大小32MBytes, 可以储存4M个时间戳。高计数率下，或测量间隔较大，需要扩大缓存大小
#使用ctypes的c_char类型
dBuffer = (c_char*32*1024*1024)()

#初始化USB环境
tDll.LibUsb_Init()

# write threshold voltage, start from 0 to 5
for ch in range(0,6):
    tDll.dacwrite(ch, 500)

# write channel offset, start from 1 to 6
for ch in range(1,7):
    tDll.setuseroffset(ch, 0)


totalbytes = 0
#写入当前目录下的tstd.bin
file = open('irrt.bin','wb')
```

```
#开始测量，同步通道为1， 200忽略
tDll.start_irrt_mmf(1,200);

for i in range(1,4):
    #开始测量，同步通道为1， 200忽略
    # tDll.start_irrt_mmf(1,200);

    #等待0.4秒
    time.sleep(0.4)

    #停止测量
    # tDll.stop_irrt_mmf();

    #读取测量数据，参数1：最大读取数据字节数为缓存的大小，参数2：复制数据的目标地址
    #返回值：实际读取的字节数
    rbytes = tDll.get_irrt_mmf_usb_data(32*1024*1024, dBuffer)
    #将缓存输入转换为byte array
    bytes2w = ctypes.string_at(dBuffer, rbytes)
    totalbytes += rbytes;
    #写入文件
    file.write(bytes2w)
    print('%d turn finished' %i)


#停止测量
tDll.stop_irrt_mmf();

# #等待0.2秒
# time.sleep(0.2)
# rbytes = tDll.get_irrt_mmf_usb_data(32*1024*1024, dBuffer)
# bytes2w = ctypes.string_at(dBuffer, rbytes)
# file.write(bytes2w)
# totalbytes += rbytes




#读取残余数据写入文件， 也可以忽略
rbytes = tDll.get_irrt_mmf_usb_data(32*1024*1024, dBuffer)
bytes2w = ctypes.string_at(dBuffer, rbytes)
file.write(bytes2w)
totalbytes += rbytes
```

```
#关闭映射文件(必须)
tDll.close_irrt_mmf()

#关闭文件
file.flush()
file.close()

print("Measurement Done! " + str(totalbytes) + " bytes are written")

#退出USB环境
    tDll.LibUsb_Exit();
```

5 附录

5.1 时间戳格式

一条时间戳记录为 64 位, 8 字节，格式如下：

[63:57]	[56:0]
无符号整型，通道号	时间戳，有符号整型，LSB 单位： ps

5.2 时间差格式

当通道为 sync 信号时：

[63:57]	[56:0]
无符号整型，通道号	时间戳，有符号整型，单位： ps

当通道不是 sync 信号时：

解析时间差文件时，每 8 个字节表示一个时间差记录，格式如下：

[63:57]	[56:0]
无符号整型，通道号	相对 sync 的时间差，ps

5.3 单光子强度格式

一条强度记录为 64 位，8 字节，格式如下：

[63:40]	[39:32]	[31:0]
对应的同步脉冲的序号	通道号	计数值

第一条记录为报头，格式如下：

[63:40]	[39:32]	[31:0]
同步信号的通道号	0x00	时间窗口长度（us）

Matlab 脚本清单

序号	文件名	功能
1	proc_histogram.m	解析实时直方图保存文件
2	proc_intense_sync.m	解析光子强度测量保存文件
3	proc_t3_64b.m	解析 t3 模式保存文件
4	proc_ts_sfp	解析万兆光口时间戳保存文件
5	proc_ts_usb	解析 USB 时间戳保存文件