TDM 800
TDM 1600

User Manual

Version 1.3

Table of Contents

Support

The success of IQD and dotfast consulting is based on tight interaction to our customers. Please don't hesitate to contact I us in case you experience any problems.

This product is subject to constant development and improvement.

You have an idea for a new feature? You would like to have a change in the software?

Please contact us at any time.

Device Features

High channel count

The device offers up to 64 inputs with a timing resolution of 15,625 ps.

The hardware supports cascading of several devices to form larger units of up to 512 inputs. They all have got the same clock and same start time. But all units have got a separate data connection. This features is currently not implemented in firmware.

Continuous timing

The time starts with the first tag and measures the time over arbitrary intervals which can be up to a year.

The long-term accuracy is only limited by the quality of the external timing reference (10 MHz input)

Using the serial connection and a GPS receiver, the internal time can be referenced to UTC.

Accurate time base (optional)

The unit uses an oven controlled oscillator (OCXO) for it's internal time base.

Frequency stability: +-500 pbb (+-50 ppb on request)

Short Term Stability 1x10^-9/second Alan Variance

Please Note:

The OCXO warm up time is 3 minutes

High rate

The maximum burst rate is 250 Mcps on each input. 1024 pulses can be accepted at the maximum burst rate on each input. That makes the theoretical burst rate 16 Gcps.

The continuous transmission rate over USB is 10 Mcps, but up to 250 Mcps can be processed using the filter function. (Only the “interesting” events are transmitted to the computer)

TDM 800

The sustained Data rate in PCIe mode can be up to 900 Mcps.

The computer software is highly optimized and capable of storing 900 Mcps to SSD array.

TDM1600

Depending on the pulse distribution, the maximum data rate is between 1600 and 1800 Gcps.

Adjustable Delays

The time delay of each channel can be adjusted to compensate for different wire lengths. The delay spans a very large range of 500 us or 500 pulses. (Whichever limit is reached first)

Sorted Output

The output data is sorted by time. This makes real time processing much easier. When the adjustable delays are used, the data is sorted by adjusted time.

PCIe P2P transfer

It is possible to stream the data to another PCIe card for further online processing. This works without CPU intervention or load on computer main memory.

Parallel processing

The unit can perform data transmission and onboard processing at the same time.

Coincidence Counting

The devices has got 64 single counter and 128 pattern counter.

The counter can be read in a synchronized fashion so all counter represent exactly the same time interval.

A pattern be any combination of all 64 inputs. Each input can be defined as “preset”, “not present” or “don't care”.

Complex pattern using photon count

The 64 inputs can be viewed as 16 groups of 4. Each group can be used as a photon counter.

This way very complex pattern can be generated e.g. “more than 2 photons in group 1 and less than 2 photons on group 2 and exactly 2 photons on group 3”

Please note: Currently coincidence counting and data streaming are not implemented in the same firmware. Currently coincidence counting is only possible using the USB connection.

Hardware

Connectors

Time-tag Inputs

The device offers up to 64 inputs. They are located on the front side of the device.

Connector	SMA
Termination	50 Ohm
Time resolution	15,625 ps
Threshold	Adjustable from –2 V to 2 V
Coupling	DC
Relevant Edge	Positive or negative

10 MHz Input

The 10 MHz can be used to increase the stability of long-term measurements by connecting a stable clock source.

Connector	BNC
Termination	1kOhm
Coupling	AC

Levels

There are two possibilities to drive the 10 MHz input:

	Level	**Termination**
Sinus	1 Vpp nominal	1 kOhm internal
Digital	Minimum level 500 mVpp	External 50 Ohm termination recommended

The input is AC coupled. For this reason it can be driven by a variety of digital signals.

When using digital signals the rise and fall time should be greater than 5 ns to avoid ringing. As long as the corresponding 10 MHz error flags do not raise, the time base can be considered as valid.

Fiducial Channel "X2"

Connector	BNC
Termination	none
Coupling	DC
Level	LVTTL (0-3.3 V)

The fiducial channel is thought to connect 1PPS or similar signals. The resolution is 1 ns and the maximum rate is 1 kHz.

Caution: Don't apply a negative voltage or a voltage larger than 3.3 V

Serial Input "X1"

This input can be used to store serial Messages of a GPS clock source. Using this input the time data can be referenced to UTC time.

Connector	BNC
Termination	none
Coupling	DC
Level	LVTTL (0-3.3 V)
Data rate	9600

Caution: Don't apply a negative voltage or a voltage larger than 3.3 V
An external converter from RS232 to LVTTL is required

Led

There are two LEDs on the front panel.

		Termination
Power	Blue	On when the unit is working.
Error	Red	On when there is an error occurs and the corresponding error flag that is not read out already.

Please Note: Before the PCIe connection is established, the error led shows that PCIe reference clock is present.

Power Supply

The unit uses the following pins of the nim connector

Signal	**Pin**	**Current**
GND	34	
+12 V	16	2.5 A

Safety of operation:

The modules are only designed for operation within a NIM crate, which is powered with a properly rated and installed power supply. In particular, in order to ensure safe operation of the logic module and any connected instruments or computers, it is critical that the NIM crate and its power supply are connected to electric ground through the power connector, as required by the local safety standards.

Ensure the module is operated vertically, and that the air free is to flow through the cooling vents in the top and bottom sides of the unit.
Never operate the module horizontally, or with covered air vents - this could lead to overheating and consequent damage.

Disclaimer:

Please note that we cannot take responsibility for any harm or damage caused to or by the logic unit or any connected devices or instruments, in the case that the unit is operated outside of a NIM crate, and/or not powered through a properly installed NIM power supply.

Hardware Requirements

NIM

The device is housed in a 4/12 NIM bin. Power supply requirement is 2.5 A on +12V.

USB Connection

* USB Cable

* Computer running Windows or Linux. TimetagExplorer is currently only running on Windows.

Please Note: In the current firmware release the USB connection is not fully supported

PCIe Connection

* Computer with at least 1 free x8 Gen3 PCIe slot.

* Windows 7 64 bit or Windows 10 64 bit

* Dolphin PXH812 Interface Card

* PCIe x8 “iPass” cable

e.g. Molex 0745460801 (2 m)

or Molex 0745460805 (5 m)

We do not recommend 7 m cables.

PCIe Save to disk

It you want to save data to disk we recommend the following setup

Computer with at least

1 free x8 Gen3 PCIe slot.

1 free x16 Gen3 PCIe slot

1x ASRock Ultra Quad M.2 Card or similar

4x Samsung SSD 970 pro 512 or similar

PCIe streaming

It is possible to stream the data to another PCIe device using PCIe P2P transfers.

To support P2P transfer, you need

* Computer with at least 2 free PCIe slots

* Intel Xeon or AMD Epyc CPU or motherboard having a dedicated PCIe routing chip.

Driver Installation

Test mode

To install the current version of the drivers, Windows has to be put in test mode to disable driver signature enforcement.

This is done like this:

Press Start->Search->type `cmd` then right-click on the result and click *Run as administrator*. In the CMD window type or copy-paste `bcdedit /set testsigning on` and press enter. Restart PC.

Hardware setup

* Install the Dolphin PXH812 interface card in a x8 or x16 PCIe slot.
* Install the TDM800 in a NIM Crate
* Connect the PCIe cable between TDM800 and the Dolphin card.

Driver installation

1.) Power NIM crate on
After a few seconds the blue POWER LED goes on.

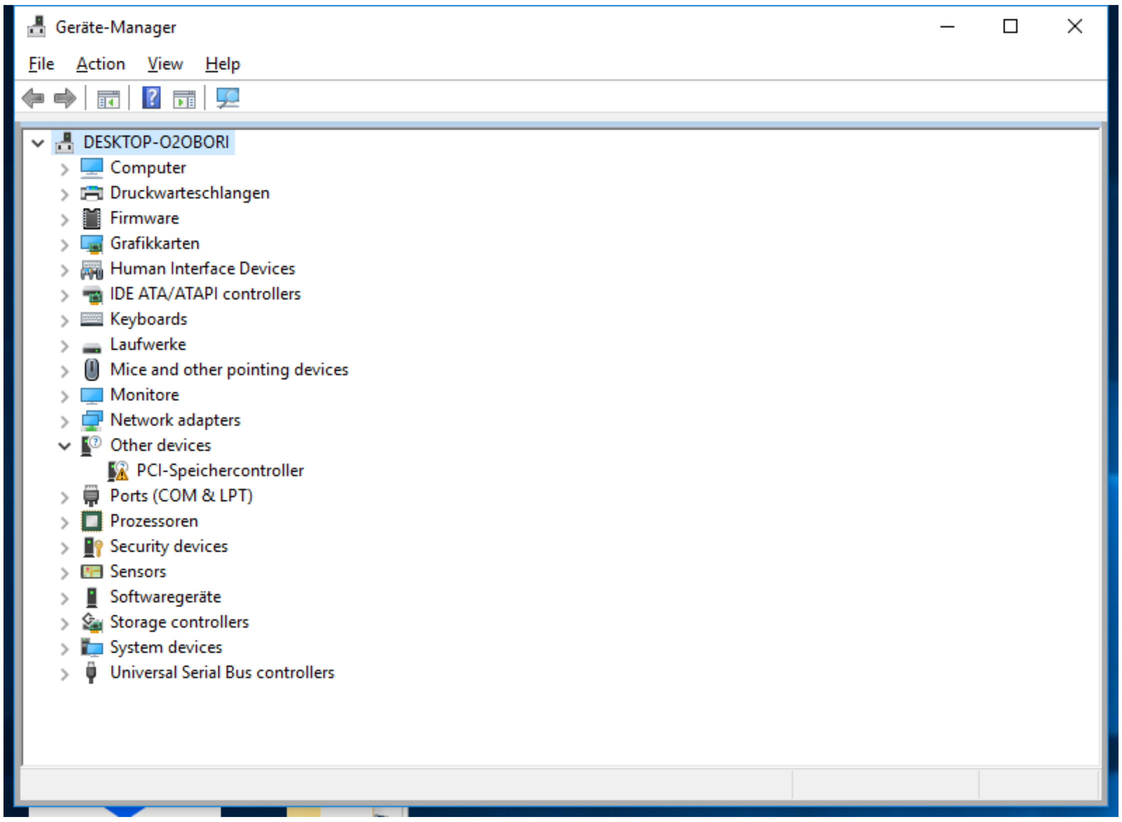
2.) Power on the computers

The red error led flashes. At this stage this is not an error; This indicates proper PCIe clock.

Please Note: Always power on the device before you power on the computer. Otherwise Windows will not recognize the device.

3.) Unzip the device driver and remember the location.

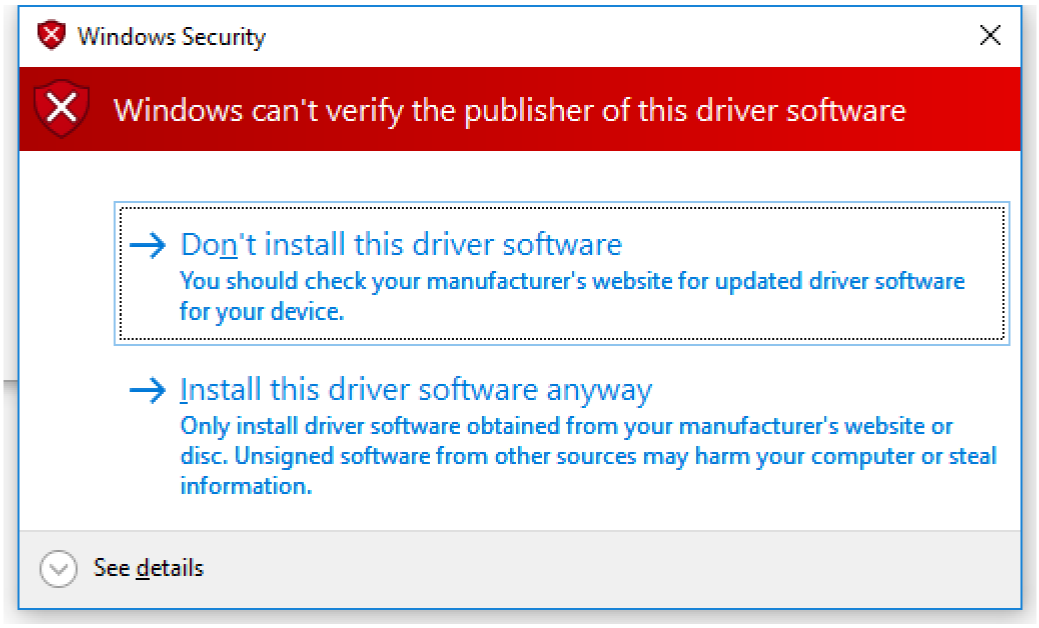
4.) Start the device manager. The device is shown as “PCI-Memory Controller”



Right click the new device, Properites, go to “Deiver” tab, click “Update Driver”. Choose “Browse my computer for driver software”.

Navitate to the location where you unzipped the driver.

Windows will complain that the driver is not signed. Click “Unstall the driver software anyways”


Windows Security

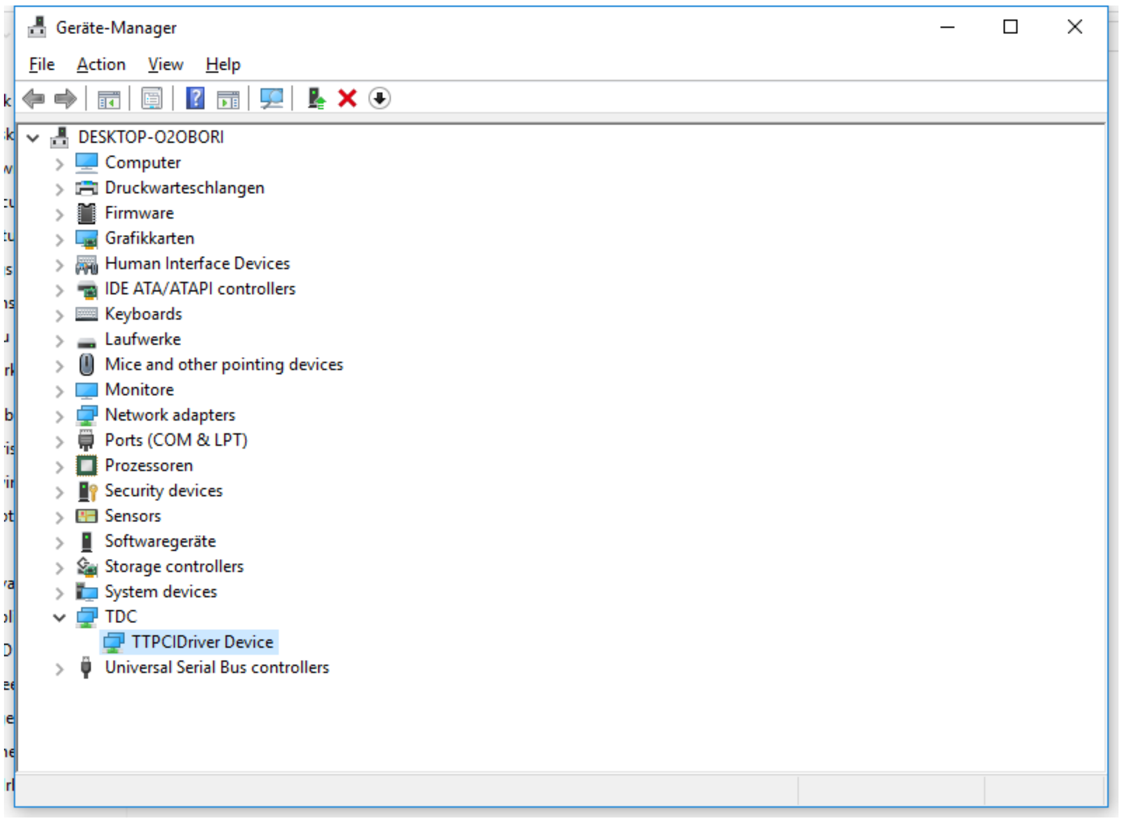
Windows can't verify the publisher of this driver software

→ Don't install this driver software
You should check your manufacturer's website for updated driver software for your device.

→ Install this driver software anyway
Only install driver software obtained from your manufacturer's website or disc. Unsigned software from other sources may harm your computer or steal information.

See details

Now the driver is successfully installed. It is listed in the category “TDC” as “TTPCIDriverDevice”



Geräte-Manager

File Action View Help

- DESKTOP-O2OBORI
  - Computer
  - Druckwarteschlangen
  - Firmware
  - Grafikkarten
  - Human Interface Devices
  - IDE ATA/ATAPI controllers
  - Keyboards
  - Laufwerke
  - Mice and other pointing devices
  - Monitore
  - Network adapters
  - Ports (COM & LPT)
  - Prozessoren
  - Security devices
  - Sensors
  - Softwaregeräte
  - Storage controllers
  - System devices
  - TDC
    - TTPCIDriver Device
  - Universal Serial Bus controllers

Setup of Timetag Explorer

Timetag Explorer is a small Windows application that helps you to get familiar with the unit without having to write your own software.

Please Note: The links in the start menu are renamed each installation. You can have several versions of TimeTag Explorer and ttInterface.dll at the same time.

Prerequisites

Before installing Timetag explorer, please take sure that the following components are installed on your computer:

- Microsoft .NET Framework 4.0 (The “Client Profile” version will do)
  It can be downloaded here:
  `www.microsoft.com/en-US/download/details.aspx?id=17718`
- Microsoft Visual C++ 2017 Redistributable Package
  It can be downloaded here:
  `https://support.microsoft.com/de-at/help/2977003/the-latest-supported-visual-c-downloads`

Using Timetag Explorer

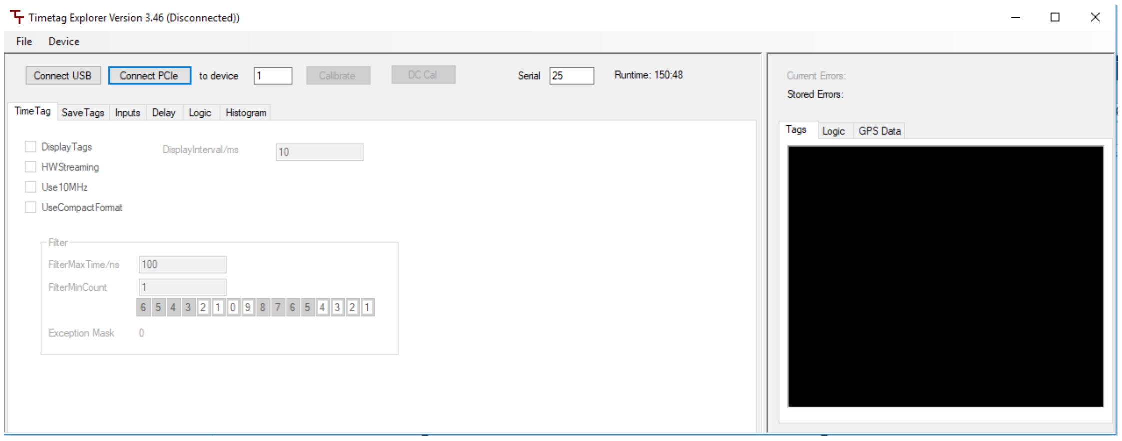
The Timetag Explorer is an easy to use application that can be used to verify the function of the device.

The functionality in the "SaveTags" tab is optimized for performance and is intended for use in the laboratory. The tabs "DisplayTags" and "Histogram" are currently not working at full speed. They are intended to do first tests and get familiar with the device.

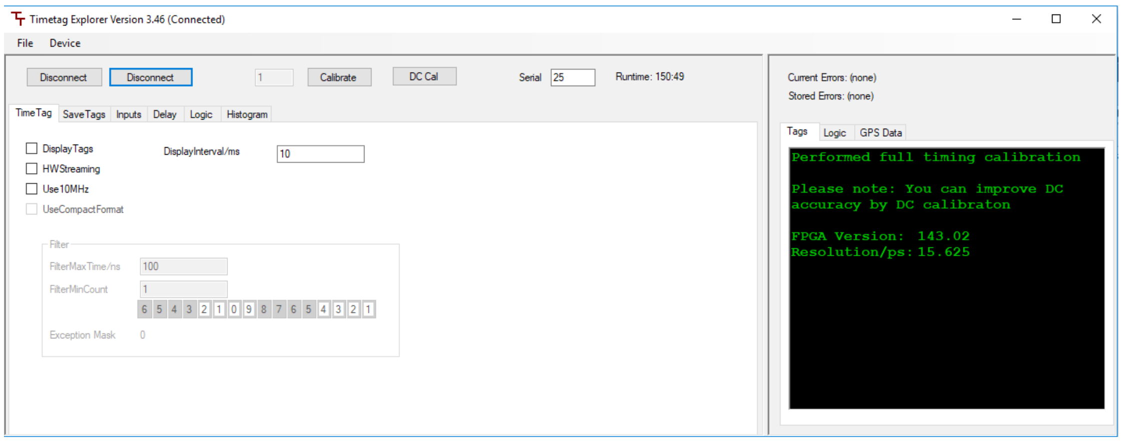
Connection

After start-up, only the “Connect USB and Connect PCIe” buttons can be selected. All other functions are disabled.

A click to the “Connect PCIe” button establishes the connection to the device and automatically stars calibration.



After successful connection the user interface looks like this:



```
Performed full timing calibration

Please note: You can improve DC
accuracy by DC calibraton

FPGA Version:  143.02
Resolution/ps: 15.625
```

Calibrate Button

The calibrate button starts the calibration sequence. It is normally not needed because the unit calibrates automatically at startup.

File Menu

The menu offers the standard “Open”, “Save” and “Save As” functions to manage parameter files. The parameter files have the extension “.timetag”.

Serial, Run time

The two fields can be used to measure the run time of the device. "serial" is an arbitrary number you can assign. "run time" is the time, the device with this serial number was connected.

The run time currently only maintained on the computer. The run time is only incremented, when the device is connected. When you use several devices, it is your responsibility to enter the correct serial number in Timetag Explorer.

The run times are written to a text file in the documents folder. When you use a device on several computers, it is you responsibility to add the values.

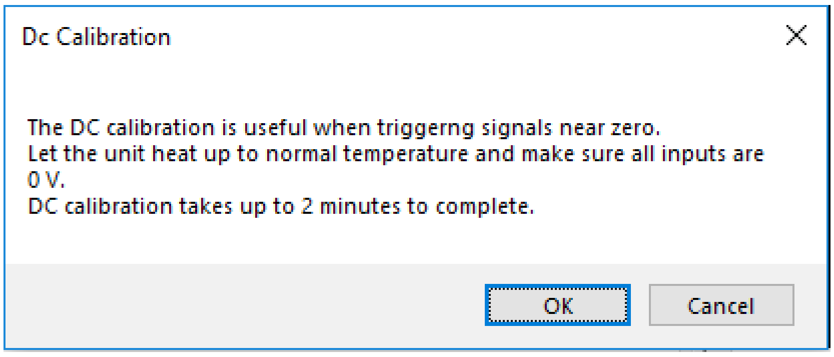
DC Calibration

DC calibration is a useful feature when triggering very small signals near zero. DC Calibration only has to be done only once for each unit.

Before starting DC calibration please make sure that all inputs have zero voltage. This is done best by disconnecting all SMA connectors.

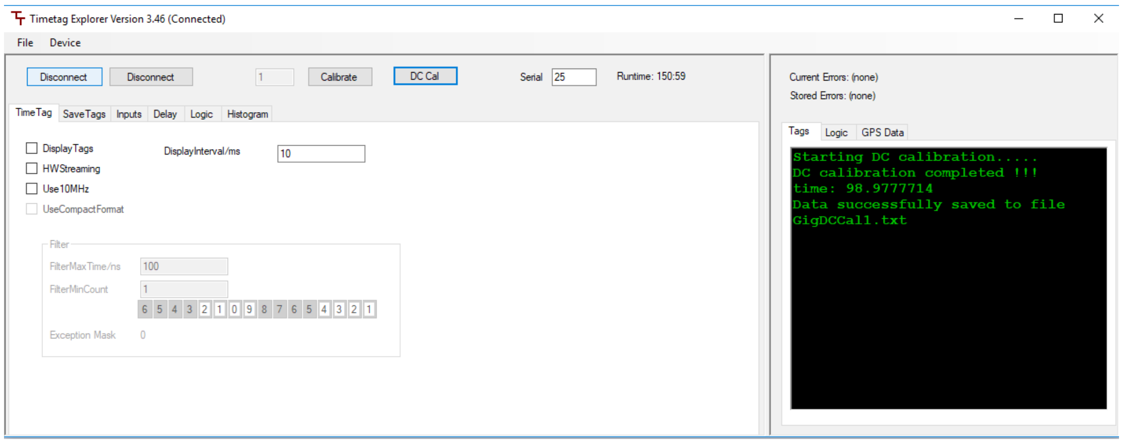
Then press the button “DC Cal”.

The following dialog appears. Press “Ok”



The DC calibration is useful when triggerng signals near zero.
Let the unit heat up to normal temperature and make sure all inputs are 0 V.
DC calibration takes up to 2 minutes to complete.

After about two minutes the calibration is completed and the calibration data is saved to disk. The DC calibration data is automatically read each time the program starts.

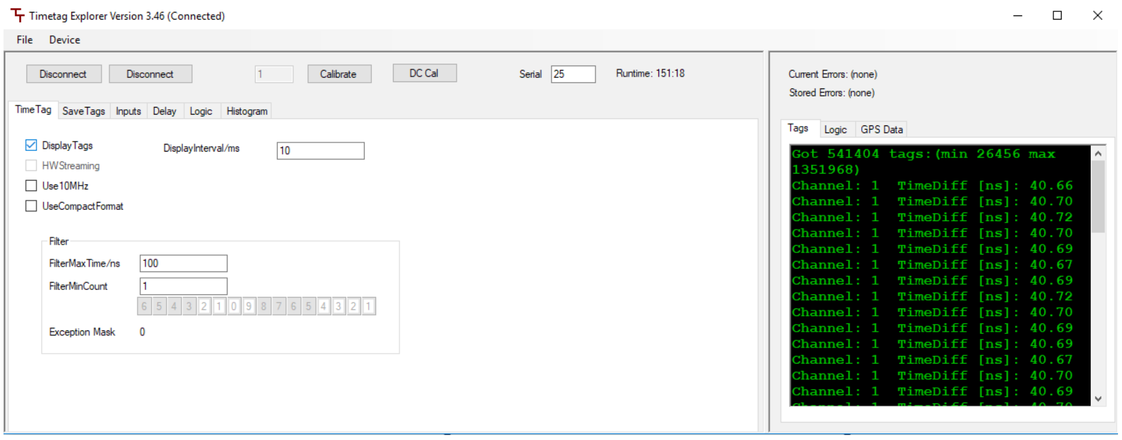


```
Starting DC calibration.....
DC calibration completed !!!
time: 98.9777714
Data successfully saved to file
GigDCCal1.txt
```

The unit number is encoded in the file name. When there are several units connected to one computer, each unit load its correct correlation data. This is true as long as you don't change the USB / PCIe topology.

Tab TimeTag

This tab contains the very basic operation modes. When selecting “Display” the tags are automatically read and the first 50 are shown in the right side of the window. Displayed are the channel number and the time difference to the previous time value.



Serial: 25
Runtime: 151:18

- [✓] DisplayTags
- [ ] HWStreaming
- [ ] Use10MHz
- [ ] UseCompactFormat

DisplayInterval/ms: 10
FilterMaxTime/ns: 100
FilterMinCount: 1
Exception Mask: 0

Current Errors: (none)
Stored Errors: (none)

```
Got 541404 tags:(min 26456 max
1351968)
Channel: 1  TimeDiff [ns]: 40.66
Channel: 1  TimeDiff [ns]: 40.70
Channel: 1  TimeDiff [ns]: 40.72
Channel: 1  TimeDiff [ns]: 40.70
Channel: 1  TimeDiff [ns]: 40.69
Channel: 1  TimeDiff [ns]: 40.67
Channel: 1  TimeDiff [ns]: 40.69
Channel: 1  TimeDiff [ns]: 40.72
Channel: 1  TimeDiff [ns]: 40.70
Channel: 1  TimeDiff [ns]: 40.69
Channel: 1  TimeDiff [ns]: 40.69
Channel: 1  TimeDiff [ns]: 40.67
Channel: 1  TimeDiff [ns]: 40.70
Channel: 1  TimeDiff [ns]: 40.69
```

Use10MHz

This checkbox selects the external timing reference. When no appropriate timing signal is connected, an error flag is shown.

HWStreaming

This mode transmits the tags to a PCIe card for real time processing.

UseCompactFormat

This mode uses a more compact data format to double the data rate. This is only supported by the TDM1600.

Caution: The gate- and filter functions are available in USB mode only.

UseLevelGate

The level gate is a very easy to use gating mode. When the input 9 is high, tags are processed normally.

When it is low, the input signals are ignored.

The checkbox “LevelGateActive” is automatically checked, when the gate signal is high.

Please note that the level gate signal has in internal jitter of 5 ns.

UseEdgeGate

The edge gate is a sophisticated gating mode. The gate opens with the active edge of the gate signal (input 8) and is open for a specified time. (Parameter GateWidth)

The position of the gate can be adjusted to compensate cable delays and similar effects.

Please note, that the gate position can be a negative and positive time interval.

FilterMaxTime
FilterMinCount

The filter can be used to reduce USB bandwidth. Tags are transmitted only, when they appear in groups. A group must have at least FilterMinCount entries. The entries of the group must be separated no more then FilterMaxTime. A group be of arbitrary length as long as there is no time gap greater than FilterMaxTime.

FilterExceptions

All Inputs marked with a „+“ are excluded from the filter. They are always transmitted. (e.g. 1pps pulse)

Caution: Using filter exceptions can cause time tags not to be transmitted in correct order. (e.g. Time differences can be negative)

ExceptionMask

This value is changes automatically when FilterExceptions is changed. This is the value to be transmitted to SetFilterExceptions().

Tab SaveTags

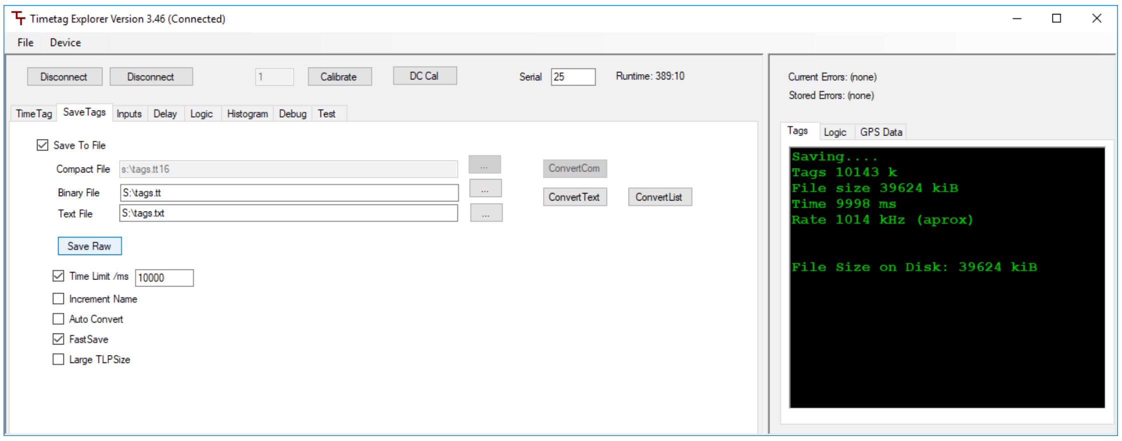
This tab can be used to store time tags to file and convert between different file formats.

The names of the binary file and the text file can be entered in the two corresponding text boxes. The path can be a relative or an absolute path. When the directory is not present, it is created.

The first file name "Compact File" is only available when connected to a TDM1600 device. It stores the file in the compact file format. The extension of this file is "tt16". Using the "Convert" Button it can be converted in the normal binary file.

Is is also possible to convert the binary file in text form. But this is only feasible for low data rates or very short times.

Clicking the “Save” button initiates the capturing of the data. Data is saved until the “Stop” button is pressed. Alternatively one can enter a time limit. In that case data is saved for a predefined amount of time, e.g. 2000 ms for 2 sec.



Serial 25 Runtime: 389:10

Current Errors: (none)
Stored Errors: (none)

☑ Save To File
Compact File s:\tags.tt16
Binary File S:\tags.tt
Text File S:\tags.txt

☑ Time Limit /ms 10000
☐ Increment Name
☐ Auto Convert
☑ FastSave
☐ Large TLPSize

```
Saving....
Tags 10143 k
File size 39624 kiB
Time 9998 ms
Rate 1014 kHz (aprox)


File Size on Disk: 39624 kiB
```

Increment Name

This can be used to store a series of files. (tags0001,tt tas0002.tt...)

Auto Convert

This mode automatically converts the file in text format after string.

FastSave

This uses a very optimized way of storing the files. This mode had some issues in earlier versions but will be the default in the future.

Large TLPSize

This has to be checked for very high rates (>800 Mcps) It works fine on most modern systems.

Convert List

The “Convert List” button automatically converts all temporary files in a certain folder. The folder and the file extension are taken from the “Temp File” text box. The text box just has to link to one temporary file. “Convert List” automatically converts all files.

Text Format

Because of file size, the text format is only recommended at low data rates.

The converted text file contains the channel number [1..64] and the corresponding time. They are separated by a tab character.

The time is stored as multiples of the internal timing resolution. The reference time is the time of the first tag.

In case of an overflow, a special tag 116 is inserted in the data stream.

Example:

```
1	0
1	6399
1	12799
1	19199
1	25598
1	31998
1	38397
1	44797
1	51197
1	57595
1	63995
1	70394
1	76794
```

Raw file format

The raw file format is a series of 32 bit integers. Each integer is divided in two fields:

Channel

The channel is encoded in the bits [31..25]. It is in the range of [0..63].

Values higher than 63 have a special meaning, see below.

Time

The low bit of the time value are encoded in the bits [24..0]. The high bits of the time value have to be reconstructed by user software.

Special channel numbers

0x7f Dummy - Please ignore

0x78 High bit - Bits [49..25] of the following tag

0x74 Overflow - At this position some data is missing

0x68 UART - Data Bits [7..0] contain serial data byte

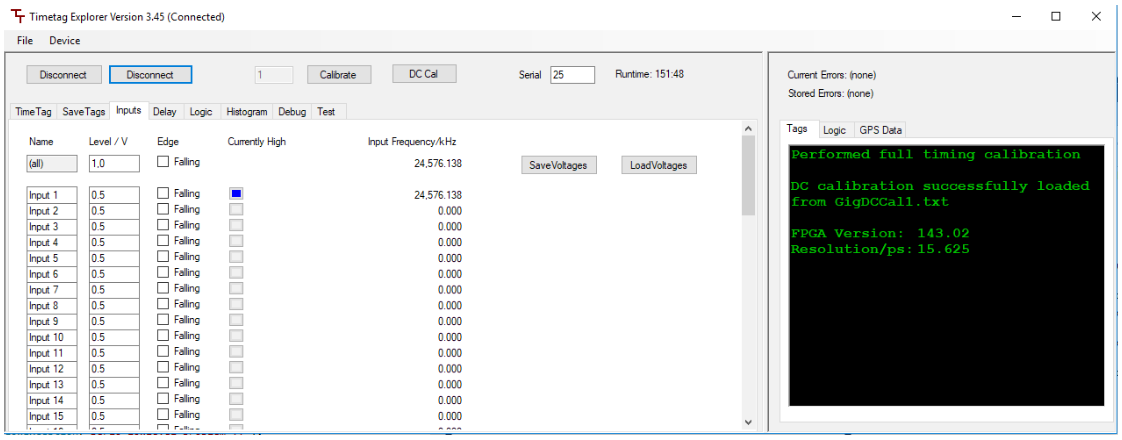
0x64 Fiducial- Data Bits [24..0] contain fiducial time

0x60 Error- Data Bits [24..0] contain error flags

0x5c Lowbit- Only used for compact data format

0x58-0x10 Reserved - Please ignore

Tab Inputs



Serial 25 Runtime: 151:48

Current Errors: (none)
Stored Errors: (none)

Name	Level / V	Edge	Currently High	Input Frequency/kHz
(all)	1,0	☐ Falling		24,576.138
Input 1	0.5	☐ Falling	■	24,576.138
Input 2	0.5	☐ Falling	☐	0.000
Input 3	0.5	☐ Falling	☐	0.000
Input 4	0.5	☐ Falling	☐	0.000
Input 5	0.5	☐ Falling	☐	0.000
Input 6	0.5	☐ Falling	☐	0.000
Input 7	0.5	☐ Falling	☐	0.000
Input 8	0.5	☐ Falling	☐	0.000
Input 9	0.5	☐ Falling	☐	0.000
Input 10	0.5	☐ Falling	☐	0.000
Input 11	0.5	☐ Falling	☐	0.000
Input 12	0.5	☐ Falling	☐	0.000
Input 13	0.5	☐ Falling	☐	0.000
Input 14	0.5	☐ Falling	☐	0.000
Input 15	0.5	☐ Falling	☐	0.000

```
Performed full timing calibration

DC calibration successfully loaded
from GigDCCal1.txt

FPGA Version:  143.02
Resolution/ps: 15.625
```

Name

In this text box a name can be assigned to each input. This is optional.

Level

This is the threshold voltage of the input.

Edge

The “Edge” checkbox toggles the relevant edge. If not selected, the positive edge is relevant.

Active

When the input has an active level, the color of this field turns to blue. The active level depends on the selected edge. When using positive edge, the active level is high.

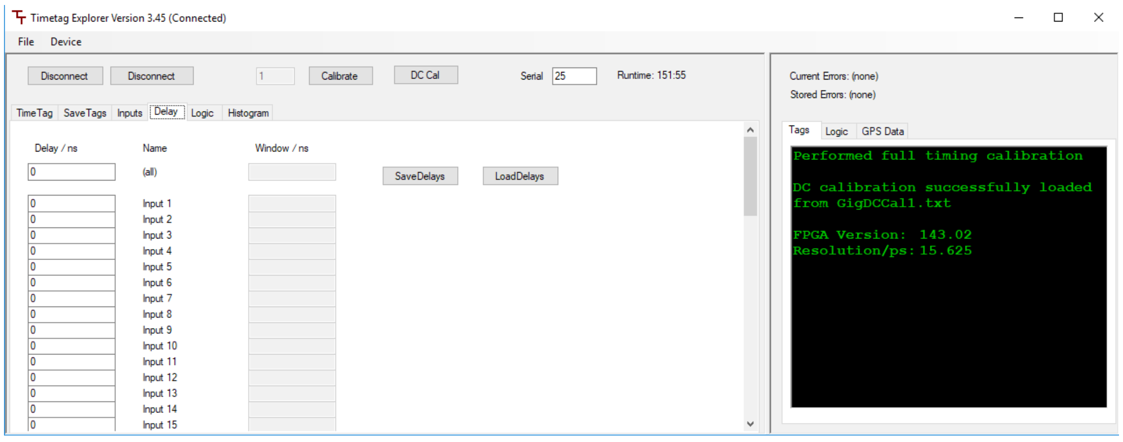
Input Frequency

This shows the single frequency of the respective input. This value is displayed regardless whether the “logic” functionality is enabled or not. Please note: The two sets of values are measured over a different time period and therefore the result can differ slightly.

The first row can be used to set the threshold voltage and the relevant edge of all inputs simultaneously. The frequency in the top row shows the total input frequency of all inputs.

The buttons "SaveVoltages" and "LoadVoltages" can be used to save / load the threshold voltages to / from a text file.

Tabs Delays

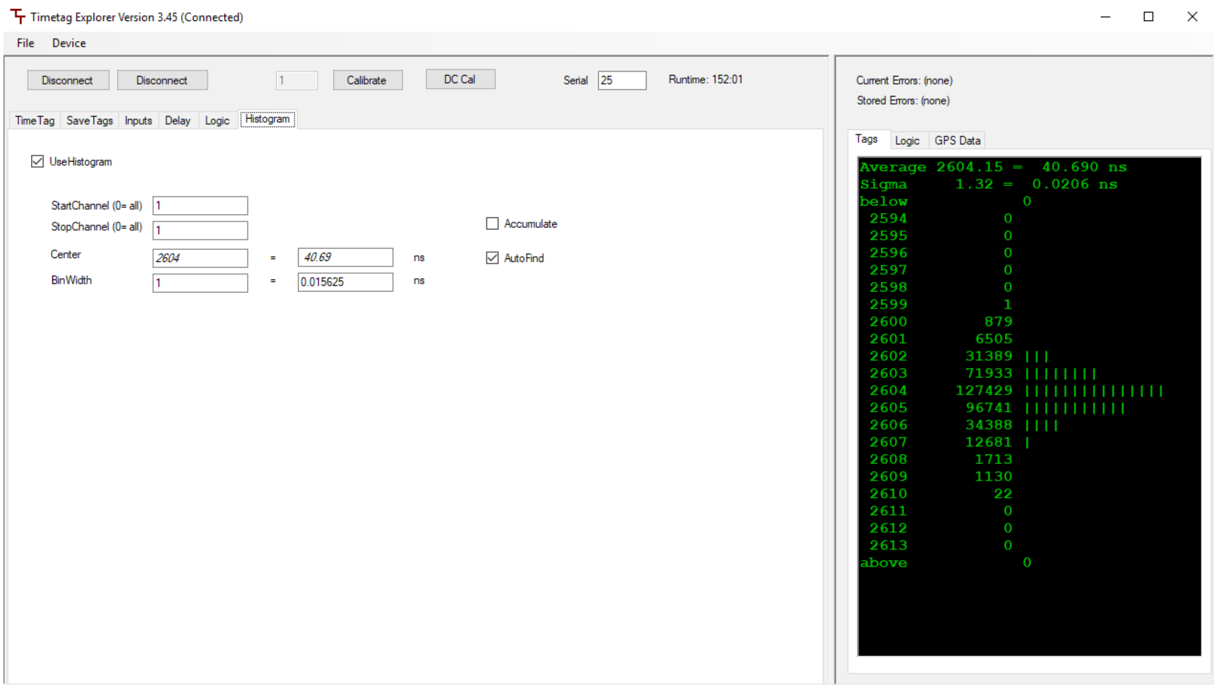


```
Performed full timing calibration

DC calibration successfully loaded
from GigDCCal1.txt

FPGA Version:  143.02
Resolution/ps: 15.625
```

Here a delay can be entered for each input. This is to compensate different cable lengths. Please note, that the internal delays of the TDM800 are currently not compensated. This compensation also has to be done here.

The buttons SaveDelays and LoadDelays can be used to save/load the delay values to/from file. The text box in the top row can be used to set all delays to the same value.

Tab Histogram



Serial 25 Runtime: 152:01

Current Errors: (none)
Stored Errors: (none)

☑ UseHistogram

StartChannel (0= all) 1
StopChannel (0= all) 1
Center 2604 = 40.69 ns
BinWidth 1 = 0.015625 ns

☐ Accumulate
☑ AutoFind

```
Average 2604.15 =  40.690 ns
Sigma      1.32 =  0.0206 ns
below            0
 2594          0
 2595          0
 2596          0
 2597          0
 2598          0
 2599          1
 2600        879
 2601       6505
 2602      31389 |||
 2603      71933 ||||||||
 2604     127429 |||||||||||||||
 2605      96741 |||||||||||
 2606      34388 ||||
 2607      12681 |
 2608       1713
 2609       1130
 2610         22
 2611          0
 2612          0
 2613          0
above            0
```

This is a very rudimentary histogram function.

Software Interface

There are two main possibilities to access the unit using software

.NET

.NET assemblies can be integrated in most programming environments, including LabView and MatLab.

The software interface is contained in the class ttInterface that is present in the file ttInterface.dll. The dll file can be found in the folder where TimeTagExplorer is installed.

ttinferface.dll uses .NET framework 4.0 which is not officially supported by older versions of LabView. But it still works because none of the 4.0 features is actually used.

To use 4.0, the following configuration file is required:

The configuration file must be placed next to `LabVIEW.exe` and must be named `LabVIEW.exe.config`. The following example instructs LabVIEW to load the CLR 4.0:

```
<configuration>
<startup useLegacyV2RuntimeActivationPolicy="true">
<supportedRuntime version="v4.0.30319"/>
</startup>
</configuration>
```

Simple -NET Program

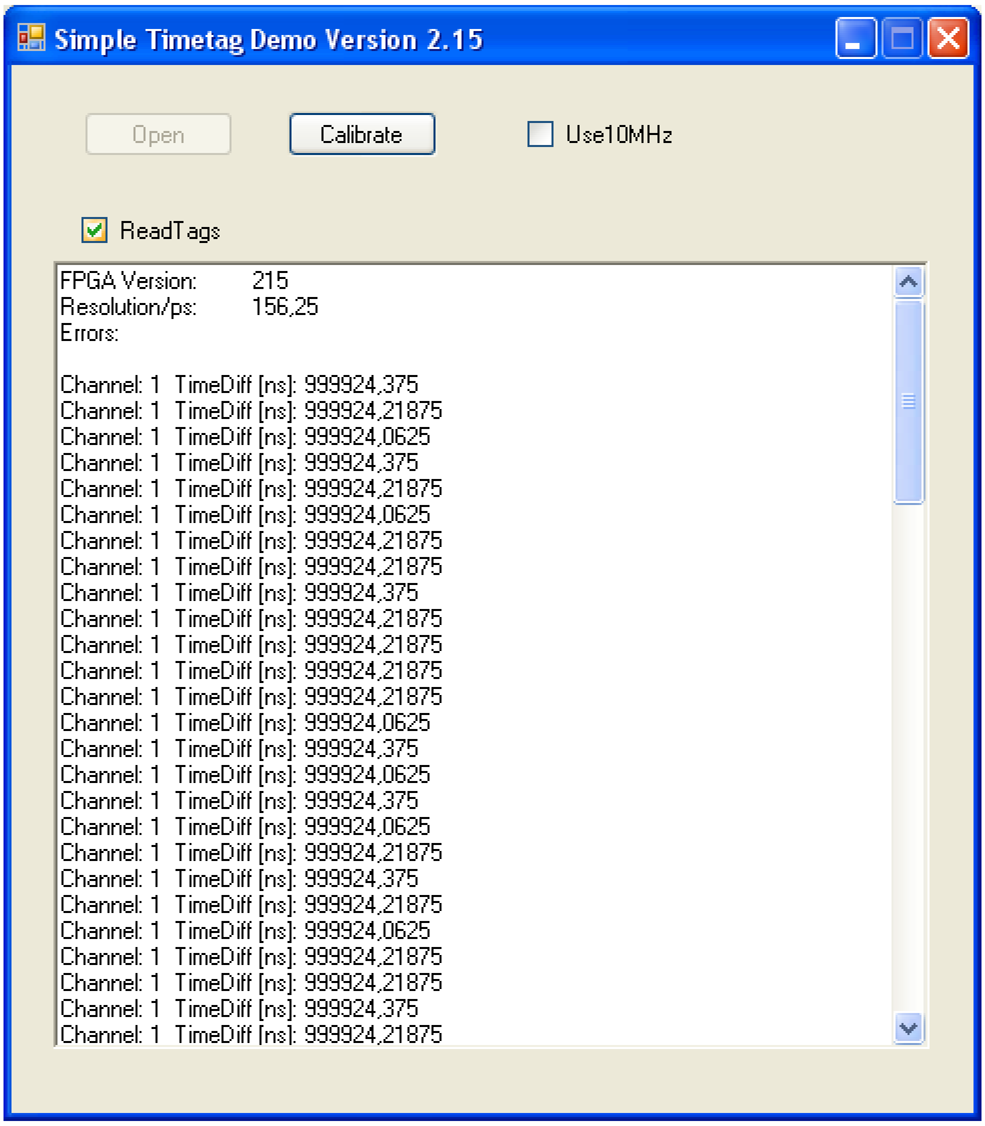
This simple program fragment shows how to use the interface:

```
ttInterface = new TTInterface(); //Create object
ttInterface.Open(1);            //Open device No 1
for (int i= 1; i<=64; i++)
 ttInterface.SetThresholdVoltage(i, 0.5);
ttInterface.StartTimetags();     //Start measurement

byte[] channels;
long[] times;
int count = ttInterface.ReadTags(channels, times);

ttInterface.StopTimetags();
ttInterface.Close();
```

On error conditions, an exception of type `TimeTag.TTException is thrown.`

How to use the .NET Demo Source Code

The demo source code is a very simple C# project that shows the basic usage of the time tag interface. The project file is “SimpleTimetagDemo.sln”. it can be opened with Microsoft Visual Studio 2017. The “Express” version of Visual Studio will also work.



```
Use10MHz: [ ]
ReadTags: [✓]

FPGA Version:       215
Resolution/ps:      156,25
Errors:

Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,0625
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,21875
Channel: 1  TimeDiff [ns]: 999924,375
Channel: 1  TimeDiff [ns]: 999924,21875
```

C++

The c++ literface is compiles using Microsoft Visual Studio 2017.
The files are contained in the folder “CTimeTag”.

File	Description
include\CTimeTag.h	General header file
include\CLogic.h	Header file for Logic special functions
Win64\CTimeTagLib.lib Win64\CTimeTagLibDebug.lib	Windows 64 bit

Simple Program Fragment

```
   CTimeTag timetag;
   timetag.OpenPCI(1);              //Open device No 1
   timetag.StartTimetags();      //Start measurement
unsigned char *chan;
long long *time;
int count= timetag.ReadTags(chan, time);

   timetag.StopTimetags();
   timetag.Close();
```

On error conditions, an exception of type `TimeTag::Exception is thrown.`

How to use the C++ Demo Source Code

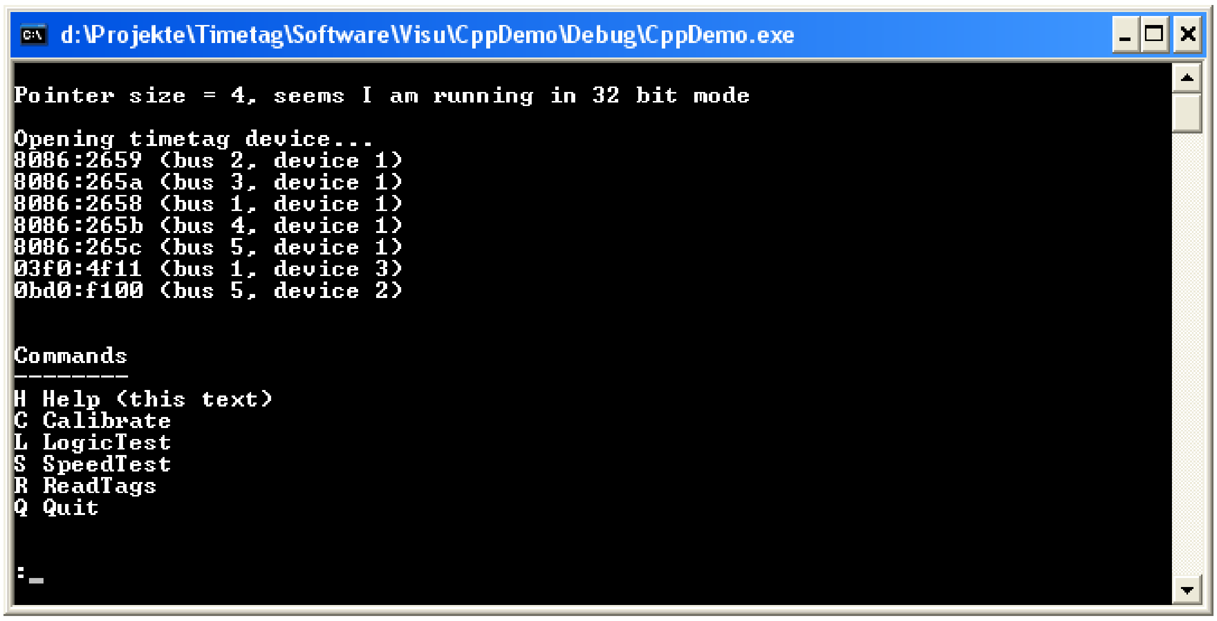
To use the demo program, just copy the two Folders “CppDemo” and “CTimeTag” into the same parent folder.

Windows:

On Windows, open CppDemo.sln using Visual Studio 2017 and compile.

Linux:

(Currently not supported)



```
Pointer size = 4, seems I am running in 32 bit mode

Opening timetag device...
8086:2659 (bus 2, device 1)
8086:265a (bus 3, device 1)
8086:2658 (bus 1, device 1)
8086:265b (bus 4, device 1)
8086:265c (bus 5, device 1)
03f0:4f11 (bus 1, device 3)
0bd0:f100 (bus 5, device 2)


Commands
--------
H Help (this text)
C Calibrate
L LogicTest
S SpeedTest
R ReadTags
Q Quit


:_
```

Basic Functions

The following description applies to all configurations of the device. They are written in C# syntax but the C++ methods have the exact same signature.

```
public void OpenPCIe(int Number)
```

Number	>=1	The parameter is used only when more than one units are connected at the same time. If only one unit is used, this value must be set to 1

This function connects to the device. It has to be called before any other function is called.

```
public void Close()
```

This function should be called before the program terminates.

```
public void Calibrate()
```

This function is optional, because the unit is calibrated automatically.

```
public long ReadErrorFlags()
```

This function returns the internal error flags. Calling this function clears the error flags in the device.

The flags are bit-encoded. Flag 0 result in a value of 1, flag 1 result in a value of 2, Flag 2 results in 4 and so on.

Flag No	**Flag Name**	
0	`DataOverflow`	An overflow in the 512 k values SRAM FIFO has been detected. The time-tag generation rate is higher than the USB /PCIe transmission rate.
1	`NegFifoOverflow`	Internal reason, should never occur
2	`PosFifoOverflow`	Internal reason, should never occur
3	`DoubleError`	One input had two pulses within the coincidence window.
4	`InputFifoOverflow`	The delay value is too large for the input frequency. Only 512 pulses can be stored.
5	`10MHzHardError`	The 10 MHz input is not connected or connected to a wrong type of signal.
6	`10MHzSoftErrorMin`	The 10 MHz input is connected, but the frequency is not 10 MHz. (Periode too small)
7		
8	`OutDoublePulse`	An output pulse was generated, while another pulse was still present on the same output. The pulse length is too long for the given rate.
9		

10		
11		
12	`10MHzSoftErrorMax`	The 10 MHz input is connected, but the frequency is not 10 MHz. (Periode too large)
13	`LevelError`	Problem in sort engine. Some data is lost, but remaining data is valid.
14	`IndexTooLate`	Internal error, frequency probably too high. Data may be corrupted.
15	`IndexErr`	Internal error, frequency probably too high. Data may be corrupted.
16	`ZeroBitErr`	Internal error, should never occour. Data may be corrupted.
17	`LargeFifoFull`	Problem in sort engine. Some data is lost, but remaining data is valid.
18	`CompactorFifoFull`	Problem in sort engine. Some data is lost, but remaining data is valid.
19	`FrameError`	Frame error on serial input
20	`FiducialDoubleErr`	Frequency on Fiducial input too high.
21	`OverrunError`	Overrun error on serial input
22	`CheckBitError`	Internal error, frequency probably too high. Data may be corrupted.
58	`HighbitError`	
59	`NoHighbit`	
60	`Serio`	Possible hardware fault connected with level adjustment
61	`DoublePipe`	Problem with USB connection, it couldn't recover
62	`PipeError`	Problem with USB connection, it could recover
63	`OutOfSequence`	This is an internal error generated in the PC software. It is generated when the tags are not sorted correctly. Plase contact factory.

```
public string GetErrorText(long flags)
```

This function translates the error flags to a short text that can be displayed on the user interface. The error flags can be obtained by GetErrorFlags() or extracted from the raw data stream. Both use the same encoding.

```
public int GetNoInputs()
```

This function returns the number of inputs installed on the device. It is used for debugging purposes only.

```
public double GetResolution()
```

This function returns the time resolution of the device. It should be used to calculate absolute time values.

```
public void SetInputThreshold(int input, double voltage)
```

Input	[1..64]	Number of the input to change.
Voltage	[-2.0..2.0]	Threshold voltage in volt

```
public void SetInversionMask(long mask)
```

This function is maintained for compatibility only. Please use `SetNegEdge()`.

```
public void SetDelay(int input, int delay)
```

All input signals can be delayed internally. This is useful to compensate external cable delays.

Input	[1..64]	Number of the input to change.
Delay	25 bit value	Delay in internal units. (See GetResolution())

```
public int GetFpgaVersion()
```

This function returns the current version of the FPGA design. It is used for debugging purposes only.

```
public Logic GetLogic()
```

This function returns the "Logic" object, that contains the Counter / Logic functions.

```
public int64 FreezeSingleCounter()
```

This function stores all the single counters synchronously. This function also returns the time between the last two calls to FreezeSingleCounter.

The time is expressed in 4 ns ticks.

Note: This function is implemented in software versions above 3.45. Please use Logic object for older versions.

`public int GetSingleCount(int input)`

This returns the number of input pulses in between the last two calls of FreezeSingleCounter. This example calculates the frequency of input 4:

```
double  time = FreezeSingleCounter();
int pulses = GetSingleCount(4);
double frequency = pulses / time;
```

Note: This function is implemented in software versions above 3.45. Please use Logic object for older versions.

`public void SetLedBrightness(int percent)`

The Brightness of the LED on the front panel can be changed.

Note: This function is implemented in software versions above 3.45 and FPGA Version 1.46.

`public void EnableInput(int inputNo, bool enable)`

This can be used to disable inputs. After startup all inputs are enabled.

`public bool ReadInputStatus(int inputNo)`

This function checks if the input is currently low or high. This can be useful while adjusting the input levels.

`public void SetNegEdge(int inputNo, bool neg)`

After startup, the unit is configured to trigger on the positive edge. By calling this function with neg= true, the corresponding input is defined as the negative edge.

This is an alternative to the function `SetInversionMask`.

`public string LoadTemporalWalkCorrection(string filename)`

When a detector triggers twice within a short time interval, the time of the second pulse tends to be off because the detector is not fully recovered yet. This can be compensated with this function. Please call factory for details.

Time Tag Readout

The current software interface is single-threaded and can support about 200-300 MTags, depending on the computer. At time of this writing, the full data rate can only be saved to disk. If you want to do real-time processing of the data, please contact factory.

At an internal FIFO overflow, the `DataOverflow` flag is set and the error led on the front panel is lit.

The following functions are implemented in the class TimetagReader. You get a handle to the TimetagReader by calling TTInterface.GetReader();

```
public void StartTimetags()
```

This function puts the device into time-tag readout mode. The time tags are written into the internal fifo.

The background thread starts to read the time tags into the RAM of the computer.

```
public void StopTimetags()
```

The tags are no longer written into internal fifo.

The background thread is stopped.

```
public bool TagsPresent()
```

This function returns true when tags are ready to read.

This function is optional.

```
public bool ReadNextTag(out byte channel, out long time)
```

This function is easier to use than the ReadTags() function but the performance is not as high.

It returns true, when a new value is present. This function is not working in Python, because Python does not support output parameters.

Pease note: Directly after calling StartTimetags() the function will return false, because the data didn't arrive in the computer yet.

```
//Actually C++ syntax here, this method is not present in .NET code
public: struct Result * ReadNextTagValue()
```

This is the very same functionality as ReadNextTag, but optimized for Python. It returns either null (= none) or a struct Result.

```
        struct Result
        {
         char Channel;
         long long Time;
        }
}
```

```
public int ReadTags(out byte[] channels, out long[] times)
(C# Syntax)
public: int ReadTags(unsigned char * channels, long long * times)
(C++ Syntax)
```

Return value: size of arrays

time[n] is the absolute time in internal units. (15.625ps). This value is returned by `GetResolution()`.

Channel [n] is the number of the corresponding input. The first input is number 1.

The “virtual channel” 117 is used for the overflow tag. The overflow tag is sent whenever a data overflow error occurs. The data overflow tag indicates, that some data is missing at this point.

When there are no tags present, this function may wait up to 300 ms before it returns a zero result. When you don’t want this behavior, use TagsPresent() to check availability first. The array is sorted by time. When two tags occur at the same time, then the tag with the smaller channel number is transmitted first.

The array is allocated by the driver software and returned to user code with an out parameter. Returning with an out parameter is similar to returning values by a pointer or reference in C / C++;

The arrays can be used until the next call of ReadTags. When the data is needed for a longer time it has to be copied to a different array.

10 MHz Input

The 10 MHz input can be used to increase the long-term stability of the device. It has to be switched on by software to be used.

```
public void Use10MHz(bool use)
```

When the 10 MHz input is switched on, but no valid signal is connected to the input, an error flag is set and the error led on the front panel is lit.

Save to file

The software interface offers a high performance interface to save data to disk. The data is saved in a compressed, binary format and can be converted to ASCII offline.

It is best explained by an example:

Saving:

```
TTInterface tti= new TTInterface();
tti.Open()
TimeTagReader r= tti.GetReader();
```

```
r.StartSaving("rawdatafile.tt");
//Wait some time
//Monitor r.SavedTags when you like
r.StopSaving();
tti.Close();
```

Converting

```
TTInterface tti= new TTInterface();
TimeTagReader r= tti->GetReader();
r.StartConverting("rawdatafile.tt", "tags.txt");
r.WaitUntilConversionFinished();
```

For converting you don't have to call Open() and you don't even have to have a unit connected.

Of course you can do the conversion right after StopSaving().

Conversion Mode

The generated file can be either text or binary. Binary mode is selected by r.`SetConversionBinary`(`true`).

The default is text mode.

Counter Functions

There are two types of counter. Single counter and pattern counter.

Single counter

There are 64 single counter, one for each input. They are working constantly, independent of all other functions. They are present in all devices (TDM800, TDM1600, Logic 64)

Pattern counter

These count special conditions that are interesting in the experiment. This conditions are called pattern and are defined using `DefinePattern` and `DefinePatternNeg` functions. For advanced cases `SetCompactorBit` and `SetCompactorCount` can also be used.

Single Counter Functions

```
public double FreezeSingleCounter(int inputNo)
```

This function takes a snapshot of the single counters so they can be read in a coherent fashion.

The function returns the time in seconds to the last call in seconds. This time can be used to accurately calculate the input frequency.

```
public int GetSingleCount(int inputNo)
```

inputNo: 1-64

This function returns the number of pulses on the corresponding inputs between the last two calls to FreezeSingleCounter.

Pattern Counter Functions

The pattern counter functions are located in the "Logic" object. In .NET It can be obtained by the ttInterface object by the call "GetLogic()". In C++ it can by obtained by the CTimeTag object.

```
public void SwitchLogicMode()
```

This function is implemented for compatibility with older Logic-16 devices. In current devices the counter and time-tag can run simultaneously.

```
public void ReadLogic()
```

Please call this function before you read any counters. The calls to ReadLogic defines the measurement interval.

This function takes a snapshot of all counter. All counter keep running in the background, so no events are lost during readout. All following functions read these snapshots. This ensures that all values correspond to the exact same time interval.

When you use ReadLogic() please do not call FreezeSingleCounter() because ReadLogic also freezes the single counter.

```
public int GetTimeCounter()
```

This function returns the time between the last two calls to ReadLogic() in 4ns increments. This function is provided for compatibility.

```
public double GetTime()
```

This function returns the time between the last two calls to ReadLogic() in seconds increments.

```
public int GetSingleCount(int inputNo)
```

inputNo: 1-64

This function returns the number of pulses on the corresponding inputs.

```
public int GetPatternCount(int patternNo)
```

patternNo: 1-128

This function returns the number of times in the corresponding pattern occurred.

```
public int DefinePattern(int patternNo, unsigned long  posMask)
```

patternNo: Number of pattern to define.

posMask: 64 bit mask of the inputs that have to be present for the pattern to be active. Please note that the index starts at 0. Input 1 is sented by 0, input 2 by bit 1 and so on.

Example: `DefinePattern (27, 3);`

This defines pattern 27 to be the "AND" gate of the first two inputs. Counter 27 will count, when input 1 and input 2 both have an active edge within the coincidence window.

Please Note:

Defining patterns takes some time. During this time, the pattern counter can be showing unexpected results. Please only call this function, when something is changed.

```
public int DefinePatternNeg(int patternNo, unsigned long  posMask,
    unsigned long  negMask)
```

patternNo: Number of pattern to define.

posMask: Bitmask of the inputs that have to be active for the pattern to count.

negMask: Bitmask of the inputs that must not be active for the pattern to count.

```
void SetWindowWidth(int window);
```

Window: 1.. 16777215

This sets the width to the coincidence window. The value is given in internal units. (15.625 ps) The width can be quite large, that is nice for testing and finding right delays. However, the window should not be so large that one input has several pulses in the window. In that case the count is not accurate and an error flag is raised.

```
void SetWindowWidthEx(int input, int window);
```

This allows different coincidence windows for different inputs. When both window functions are called, `SetWindowWidth` has to be called first because it just sets the width of all inputs.

Bitcount Feature

The unit allows arbitrary logic functions for groups of 4 neighboring channels. (1-4, 5-8, 9-12 ..) This can be used to define pattern in terms of number of photons. For historic reasons, this unit is called "compactor".

This is an advanced feature that is not needed for normal operation.

Important:

The compactor functions have to be called before any call to `DefinePattern` or `DefinePatternNeg`.

To define the compactor memory you have two options. This is the "photon counting memory" (I called it "compactor" because it used to reduce the 64 inputs to 32.)

```
void SetCompactorCount (int input, int bitCount);
```

Bitcount is the number of photons in the group, 1-4

Bitcount "-1" means don't use counter function, use the input as it is. That's the default value. (Corresponding to the "+" symbol in TimeTagExplorer)

```
void SetCompactorBit (int input, int index, bool value);
```

Using this function you have full control over the memory. Please contact factory for details.

input: 1-64

index: 0-15 memory address

value: memory bit

Known Problems

Problem	**Solution**
All devices	
The different delay time of the individual inputs is currently not compensated. These delays can change when the units are exchanged or die firmware is updated	Compensate the internal delays together with you cables and your experiment.
The unit has issues with very low frequencies. (< 120 cps)	Connect a 200 Hz dummy signal to one of the inputs.
TDM1600 only	
Data can be unreliable when the inputs frequency is below 250 kHz	Please use the compact data format only for high speed data
The compact data format loses the absolute time base when an overflow occurs. Local time differences are correct, but differences spanning the overflow are wrong.	Overflow positions are marked in the data stream.

Revision History

Date	Document Version	FPGA Version	Dll Version	Remark
2019-10-09	0.1			Initial revision
2020-02-03	0.9	1.46	3.46	Updated software interface Updated timetag explorer
2020-03-12	1.0	1.46	3.46	TDM1600 Known Problems
2021-02-15	1.1		3.47	Error flags 64 bit Clarification on error flags
2021-07-16	1.2			Added logic functions
2022-04-05	1.3	1.49	3.56	Updated logic functions Added: EnableInput, SetNetEdge, ReadInputStatus, LoadTemporalWalkCorrection