www.qutools.com
info@qutools.com

quTAU

Time-to-Digital Converter

quTAU quPSI Manual V4.0

quTAU/quPSI Manual V4.0 (March 1, 2016)

Contents

1 Quickstart Guide **4**

2 Technical Information **5**

3 Hardware Concept **5**

4 General Information **7**
4.1 Bin width/Differential Nonlinearity . . . 7
4.2 Time tags bit width . . . 7
4.3 Software Packages . . . 7
4.4 Linux Support . . . 8

5 Windows Installation **8**

6 Firmware Update **8**

7 Software **10**
7.1 Concept . . . 10
7.2 Daisy . . . 11
7.3 quTAU GUI . . . 11
7.4 Command Line Interface . . . 13
7.5 Using the quTAU with your own software (Library functions) . . . 13

8 Input Parameter Settings **16**
8.1 Software Realizations . . . 16
8.2 DLL Usage . . . 18

9 (Coincidence) Counting **20**
9.1 Integration Time . . . 20
9.2 Coincidence Time Window . . . 20
9.3 Realizations . . . 20
9.4 DLL Usage . . . 23

10 Time Stamping **24**
10.1 Writing and loading Time Stamps . . . 24
10.2 Software Realizations . . . 25
10.3 DLL Usage . . . 26

11 Histograms **28**
11.1 Start-Stop Histograms 28
11.2 Start-Multistop Histograms: Lifetime Measurements 30
11.3 Correlation Measurements: HBT 35

12 Simulation and Testing: Demo Mode **40**
12.1 quTAU GUI Simulation 40
12.2 DLL Usage 40

13 Differences using a quPSI **41**
13.1 quTAU GUI 41
13.2 DLL Usage 41

1 Quickstart Guide

For a quick start, first install the quTAU software with the default options (see Section 5). The necessary device drivers, different GUIs for the most common tasks, and a user library – complete with examples – to support user written programs will be installed. The mostly self-explanatory quTAU GUI is accessible via the Start Menu under `quTAU/quTAU UI`. It can already be tested and explored even without a quTAU device due to the integrated demo functionality.

After connecting the quTAU device over USB2.0, please check your firmware version with the provided update utility (in the Start Menu under `quTAU/Tools/Firmware Update Utility`, see Section 6 for details). Please make sure the device will not loose power or is switched off while flashing the firmware! If everything is in order, you can start using the GUI to record and analyze data or write your own program using the functions included in the `tdcbase.dll` DLL and the LabView™ VIs.

The device has to be initialized using `TDC_init` and de-initialized with `TDC_deInit`, everything else depends on the task at hand. A complete list of all functions the DLL provides and information about how to call these functions is included as doxygen generated documentation in `userlib/doc/index.html`. You can also find descriptions and examples for the most common tasks in this manual. Working LabView™ Examples are included in the `userlib/labviewXX/examples` folder in the installation directoy, C examples can be found in the `userlib/src` Folder.

Please take note of the information about histogram binning and time tagging in Section 4.

2 Technical Information

The quTAU is an 8-channel time-to-digital converter with a time bin of 81 ps. The arrival time of incoming signals is recorded, pre-processed (count rates are computed) and transferred to a PC via USB2.0 for further analysis. The user can read out raw time tags as well as calculated start-stop histograms.

The basic quTAU model can optionally be upgraded with 3 distinct extensions: The **input hardware** extension, enabling to specify input parameters for each channel individually, and **two software extensions** for a straightforward analysis of Lifetime and Hanbury-Brown and Twiss measurements, respectively. Please note that all three upgrades are already integrated in the quTAU (H+) model.

The key features of the quTAU device are:

- 8 channels
- High timing resolution (bin size 81 ps)
- High event rates
- Coincidence counting integrated
- USB2.0 interface
- Easy-to-use

The device can be used for a variety of Applications:

- Time correlated single photon counting (TCSPC)
- Fluorescence lifetime imaging
- Quantum information experiments
- LIDAR
- High energy/accelerator physics
- High precision time measurements

3 Hardware Concept

The hardware of the quTAU mainly consists of an ASIC (converting the incoming signals in all 8 channels into time tags) and an FPGA (sorting the time tags, detecting coincidences in the first 4 channels and compressing the data for transfer via USB2.0). The output is limited only by the speed of the USB2.0 connection: Up to 5 million events per seconds can be transferred.

Table 1: **Input Specifications.**

Channels		8
Bin width (ps)		81
Input pulse high level (V)	Min	2.4
	Max	5
Input pulse width (ns)	Min	4
Input pulse separation (ns)	Min	5.5
Input Impedance (Ω)	(LV)TTL	50/5000
Max event rates (Mevents/s)	1ch counting	10
	8ch counting	25
	8ch time tags	3
Software Delay (ns)	Min	-50
	Max	50

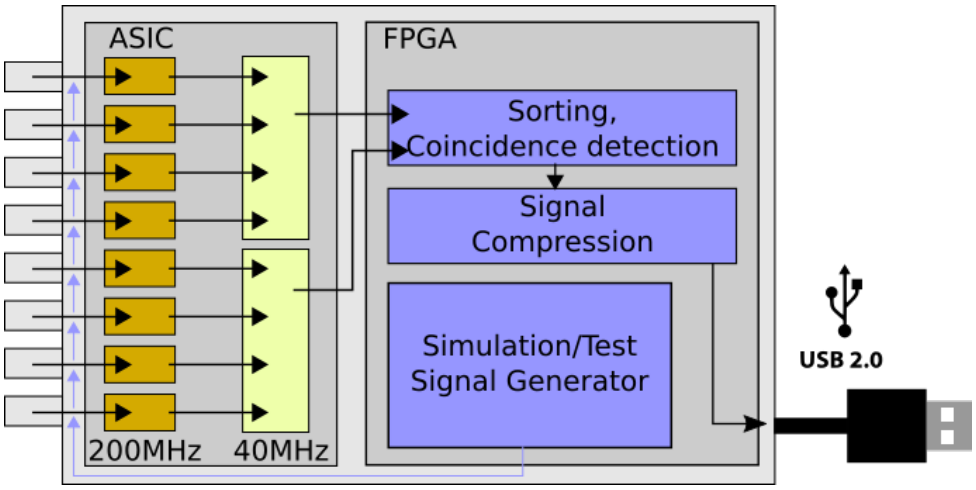

Figure 1: **Hardware concept of the quTAU.** An ASIC converts the incoming signals into time tags, which are then analyzed by an FPGA and transferred to a PC via USB2.0.

4 General Information

4.1 Bin width/Differential Nonlinearity

The width of adjacent inherent time bins (of $81\,\mathrm{ps}$) is not constant, but rather alternating. After each narrower one, there follows a wider one. This is a characteristic feauture of the TDC chip in the device and can lead to effects like the time difference histogram shown in Figure 2.

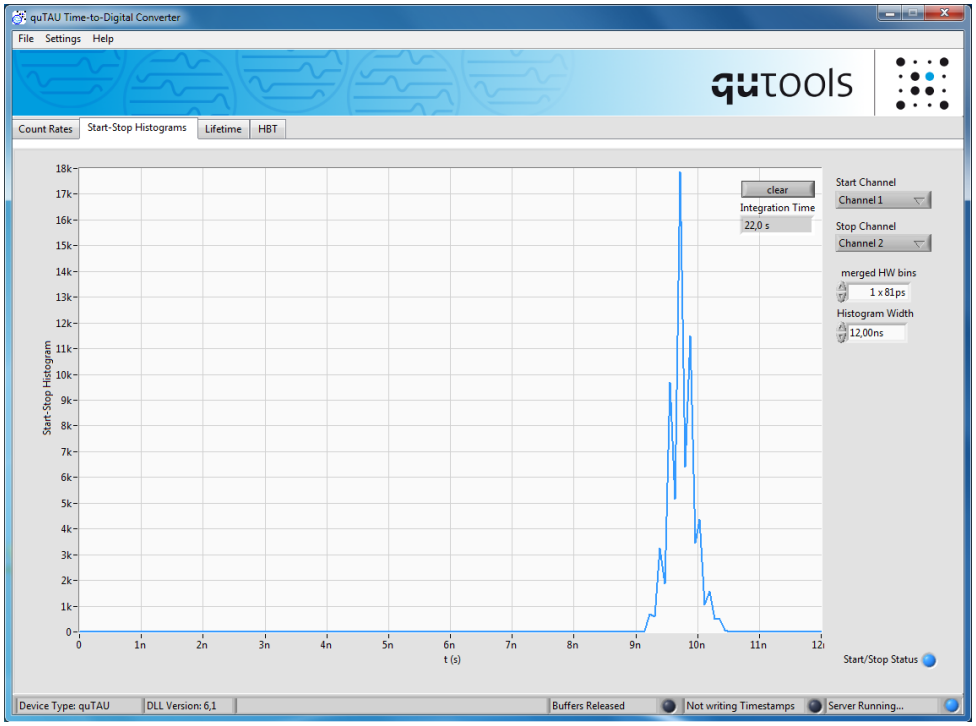

Figure 2: **Result of alternating width of time bins in a time difference histogram** when using an odd number of bins for histogram binning. Instead of a gaussian distribution, the histogramm shows a comb structure due to the differently wide time bins. This effect vanishes for even numbers of summed bins.

4.2 Time tags bit width

The internal size of time tags is $56\,\mathrm{bit}$. Therefore, they will overflow after $(2^{56}-1)\cdot 81\,\mathrm{ps} = 5836665.12\,\mathrm{s} = 67.55\,\mathrm{d}$.

4.3 Software Packages

The latest drivers, firmware and software can always be downloaded from our website: `www.qutools.com/download/list.php`

4.4 Linux Support

The quTAU offers Linux support for current major distributions. Although this manual focuses on Windows users and the quTAU GUI was designed for Windows, most of the concepts are identical or very similar. If you have specific questions or problems using the quTAU with Linux, please contact us directly.

5 Windows Installation

- Supply power to the quTAU device via a cattle plug, connect it to a PC using a USB2.0 cable and switch it on.
- Use the provided installer and follow the instructions to install all necessary components.

6 Firmware Update

Attention: Naturally, firmware updates are a delicate process and can potentially harm the device if done incorrectly. Do so only when it's advised by the manufacturer. In that case, please read and follow these instructions carefully.

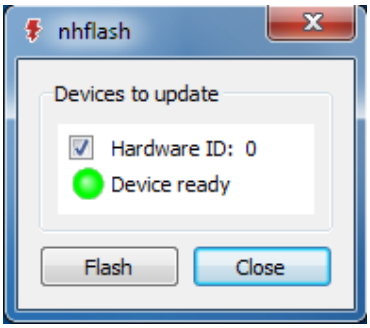

Figure 3: **The Firmware Update Utility NHFlash is ready to start the update.**

When a firmware update is required, please use the program `daisy/nhflash.exe` (`Firmware Update Utility` in the Start Menu) to do so. The current firmware is located in the `daisy/firmware` directory and consists of three files `qutau_core.bit`, `qutau_appl.bit` and `qutau_dsp.bit`. They contain firmware for the core FPGA (infrastructure), the application FPGA (handling the TDC chip) and the DSP of the quTAU device, respectively. On startup, the update utility checks whether all three files are in place and will tell you if not. Please stick closely to the following steps:

1. If the device is running, please switch it off.
2. If any quTAU software is still running, please close it.

3. Turn on the device.
4. Start `nhflash.exe` from the start menu or the `daisy` directory. After some time, the LED should become green and the text "Hardware ID: 0" (or similar) should be displayed (see Figure 3).
5. If all of this is satisfactory, please press "Flash". The green LED should start flashing until it is finished. At the end of the process, the LED should be green again and the text close to it should read "Update complete". Make absolutely sure the device is not shut down or losing power during this step!
6. Close the firmware update utility.
7. Turn the quTAU device off (and possibly on again). If you don't do this, the old firmware will still be in effect until the next restart.

7 Software

Software for the most common tasks is already provided by qutools GmbH. Here, we only take a quick look at these solutions and their advantages. A more thorough explanation will follow in the description of the different tasks.

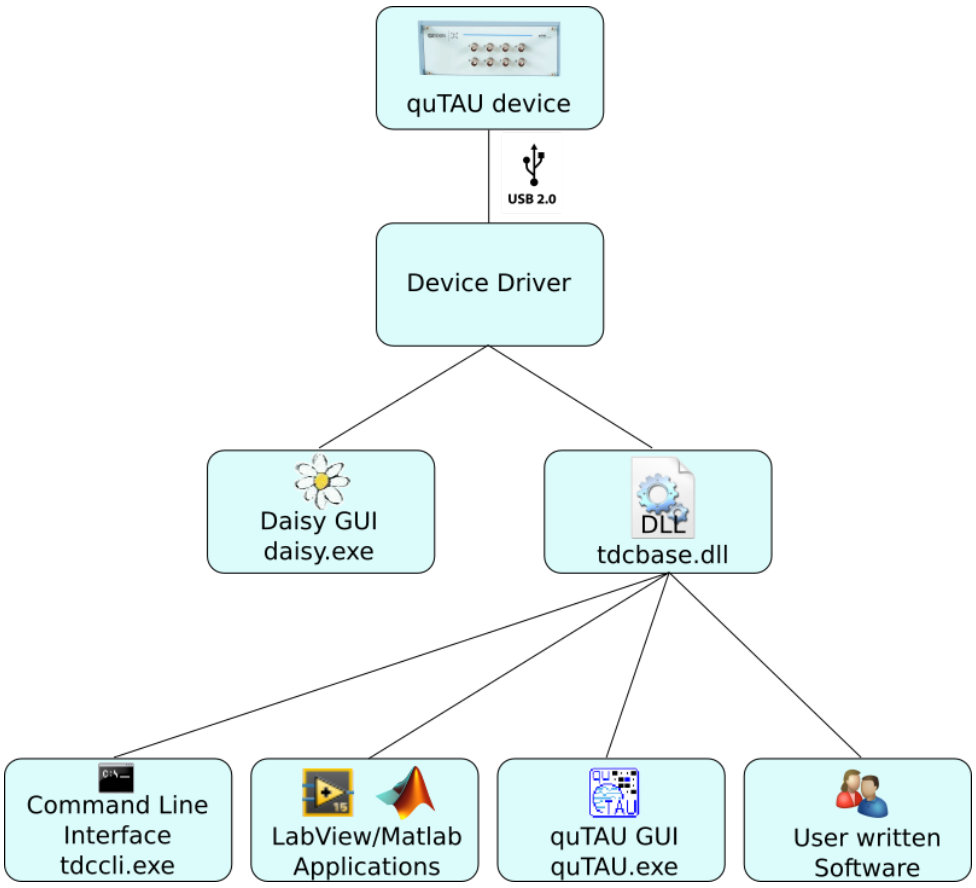

Figure 4: **Software concept of the quTAU.** The quTAU device is connected via USB2.0 with the PC. Except the Daisy GUI, all applications are using the DLL `tdcbase.dll` to communicate with the device driver.

7.1 Concept

The quTAU software concept is depicted in Figure 4. The device is connected via USB2.0, so the first piece of software handling it on the PC side is the USB device driver. The device driver then talks to directly to one (!) of the client programs, so either to the daisy GUI client or to the user DLL. The DLL then communicates with the command line interface, the quTAU GUI, LabVIEW™ VIs or other user written programs.

7.2 Daisy

The native GUI Daisy is now mostly obsolete, since practically all features are also available in the new LabVIEW™ based quTAU GUI. It is still included in this software package, but will be no longer maintained from now on. It does not use the `tdcbase.dll` DLL. The opening screen is shown in Figure 5, the different features are simply accessed by clicking on the different tabs. Double-clicking a tab decouples it from the main window, making it possible to view multiple tabs at once.

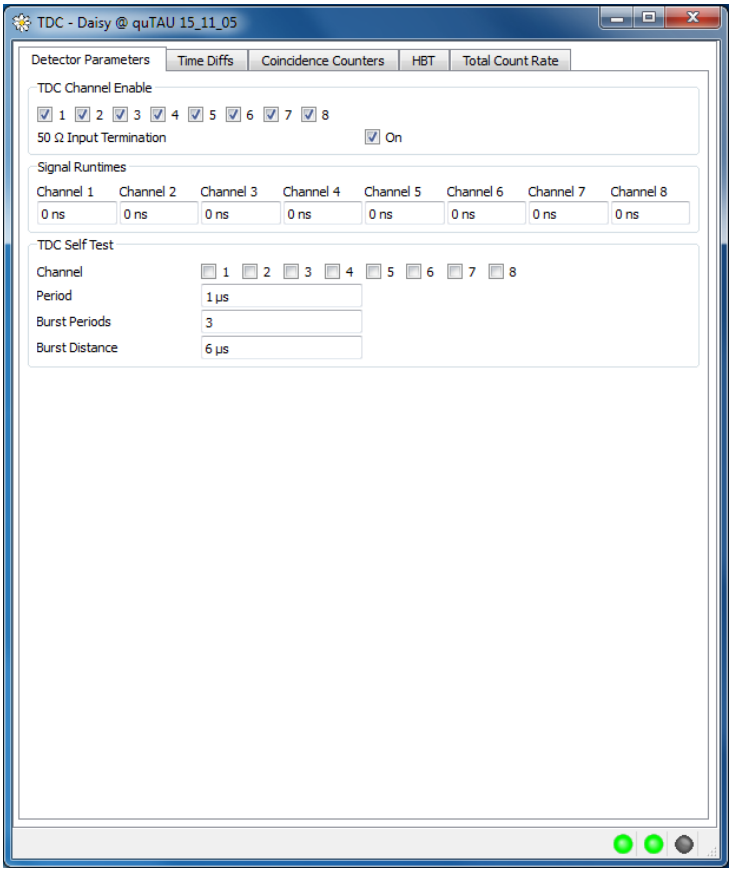

Figure 5: **The opening screen of the Daisy GUI.** Different tasks can be accessed by the different tabs at the top. Please note that this window will display additional options for upgraded quTAU devices, so the appearance varies depending on the used device.

7.3 quTAU GUI

The quTAU GUI is designed to easily perform the most common tasks like showing the count rate and coincidence rate trend for each channel, saving timestamps to a file, and displaying different histograms like start-(multi)stop histograms or cross correlation histograms ($g^{(2)}$ functions). The opening screen is shown in Figure 6. A convenient menu and the different tabs allow a self-explanatory usage of the program.

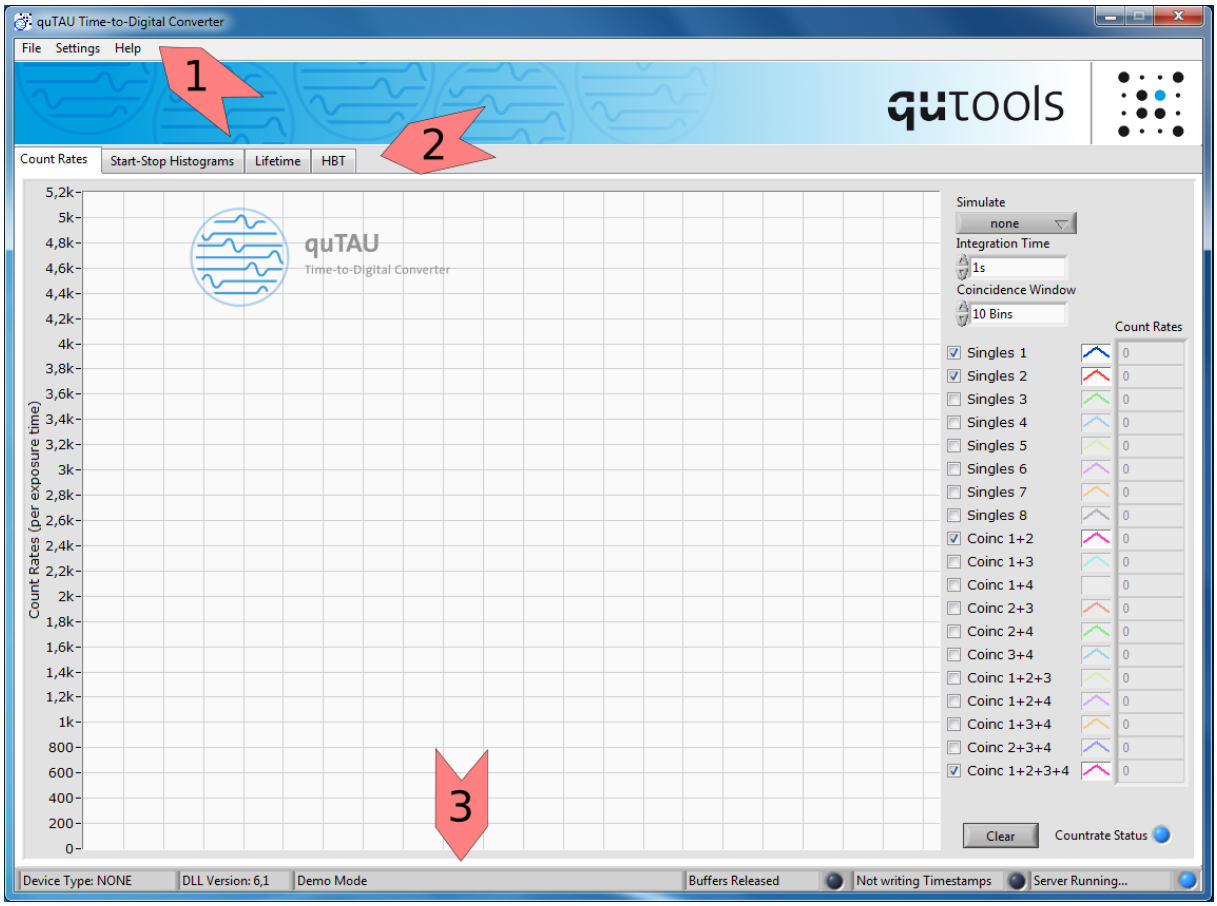

Figure 6: **The opening screen of the quTAU GUI.** By using the menu (1), timestamps can be saved to a file, a settings dialog can be opened and additional information can be displayed. The tabs (2) allow access to the different features of the device. A status bar (3) at the bottom shows additional information and the state of software and device.

7.4 Command Line Interface

The command line interface can be used for some simple tasks as a one-time readout of the count rates, writing time stamps to a file or saving histogram data. It is mostly intended as an example, though. To use it, open a Windows command prompt (Press "Start", type "cmd" and press Enter), go to the directory `userlib/daisy/` and call `tdccli(.exe)`. Use the parameter `-h` for additional information.

7.5 Using the quTAU with your own software (Library functions)

In order to enable customers to write their own software for using the time-to-digital converter quTAU, qutools GmbH provides a DLL containing all functions needed to communicate with the device. For use with LabVIEW™, a wrapper VI is provided for each of these functions (`userlib/labviewXX/lowlevelvis`). Please note that the name of the VI is equal to the DLL function name. Examples in C (`userlib/src`) and LabVIEW™ (`userlib/labviewXX/examples`) show how parts of a program might look like.

7.5.1 The qutools `tdcbase.dll` DLL/lib

The DLL `tdcbase.dll` is provided by qutools GmbH to enable customers to use the time to digital converter quTAU in their own software projects. It is available for MS Windows as well as for Linux and in versions for 32 bit and 64 bit architectures. You can download the latest version from the qutools homepage. It is also used by both the LabVIEW™ application and the command line interface, see above.

A complete list of all functions the DLL provides and information about how to call these functions is included as doxygen generated documentation in `userlib/doc/index.html`. Additionally, there will be examples of how to combine the functions to achieve a certain task. To visualize this, simplified LabVIEW™ block diagrams will be used. Please remember that the (shown) names of the used VIs are equal to the DLL function names.

7.5.2 LabVIEW™ Integration

For integration of the quTAU in LabVIEW™ using the DLL, wrapper VIs are provided at `/userlib/labviewXX/lowlevelvis/`. Each of these low level VIs can be used to call the respective DLL function. When using the low level VIs, the following facts are helpful to know:

1. The DLL file `tdcbase.dll` must be found by the LabVIEW™ program. For this purpose, the first DLL call triggers a search for the DLL file. It is expected in the application directory or any of its parents. Additionally, it can be placed in a sub folder `lib`, `lib32` or `lib64`. If the DLL is not found, the user is prompted to provide the path. In order to avoid this prompt, please place the DLL as described.

2. All low level VIs have error in- and outputs. This is convenient to determine their temporal sequence by wiring the error line from one to the next. It is important to understand that the VIs are designed to do nothing in case they are called already with an error. The error output contains the return code as well as the corresponding error message.

7.5.3 Device Initialization

Table 2: **Examples: Device Initialization**

LabView Example	`/userlib/labviewXX/examples/demo_mode.vi`
C Example	–

In order to communicate with the quTAU device, the DLL has first to be initialized using `TDC_init`. In case the device is not found or available, the init function returns the error code "2": "No connection was established". Nevertheless, the DLL will still be initialized in DEMO mode. This allows for testing and offline use of the DLL functions. On Labview, make sure to catch this error, otherwise all subsequent DLL calls will not be executed. An example is shown in figure 7.

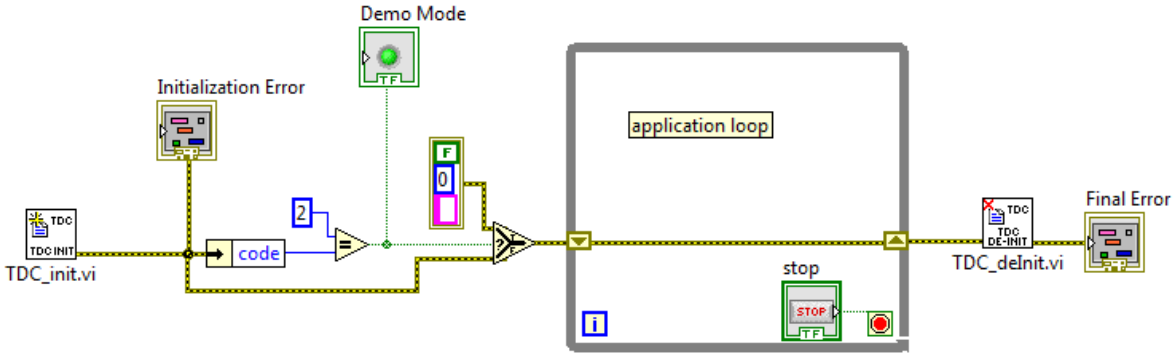

Figure 7: **Initialization,** see example `demo_mode.vi`. The return code "2" signals "No connection was established" and the DLL automatically switches to demo mode.

Necessary functions for the device initialization:
tdcbase.h:

int	**TDC_init** (int deviceId)

Additional functions related to the device initialization:
tdcbase.h:

double	**TDC_getVersion** ()
const char *	**TDC_perror** (int rc)
double	**TDC_getTimebase** ()
TDC_DevType	**TDC_getDevType** ()
BIn32	**TDC_checkFeatureHbt** ()
BIn32	**TDC_checkFeatureLifeTime** ()
int	**TDC_enableChannels** (Int32 channelMask)

7.5.4 Device Deinitialization

Table 3: **Examples: Device Deinitialization**

LabView Example	`/userlib/labviewXX/examples/demo_mode.vi`
C Example	-

Please make sure `TDC_deInit` is always called at the end of your programs. This is necessary to terminate the device properly and to enable a reinitialization for the next program start.

Necessary functions for the device DE-initialization:

tdcbase.h:

int **TDC_deInit** ()

Additional functions related to the device DE-initialization:

tdchbt.h:

void **TDC_releaseHbtFunction** (TDC_HbtFunction *fct)

tdclifetime.h:

void **TDC_releaseLftFunction** (TDC_LftFunction *fct)

8 Input Parameter Settings

You might want to change the default configuration of the digital inputs, like the trigger level, termination, or rising/falling edge selection of the input channels. Depending on your device (with or without hardware extension) not all options might be available. Table 4 gives an overview.

Table 4: **Digital input configuring.** Available features of the different models.

	quTAU	quTAU (H)
trigger level	fixed	adjustable per channel
$50\,\Omega$ termination on/off	globally	per channel
rising/falling edge selection	-	per channel
channel delay	-	per channel

- Trigger level: necessary threshold voltage to trigger an event.
- Termination: switches the termination of the input channels between $50\,\Omega$ and $5\,\mathrm{k}\Omega$.
- Edge selection: determines whether the event is triggered at the rising or the falling edge of a signal exceeding the voltage threshold.
- Channel delay: Sets additional delays for the input channels. Please note that this currently doesn't influence coincidence counting. It affects all histograms, though.

Please note that these settings are saved in the device and therefore persistent over the different software realizations. A restart of the device, however, will reset all settings to the default state.

8.1 Software Realizations

Input parameters can be set in the Daisy GUI as well as in the quTAU GUI. There are some presets available in both cases, to quickly change all parameters to commonly used values.

8.1.1 Daisy

All settings can be accessed in the "Detector Parameters" tab, see Figure 8.

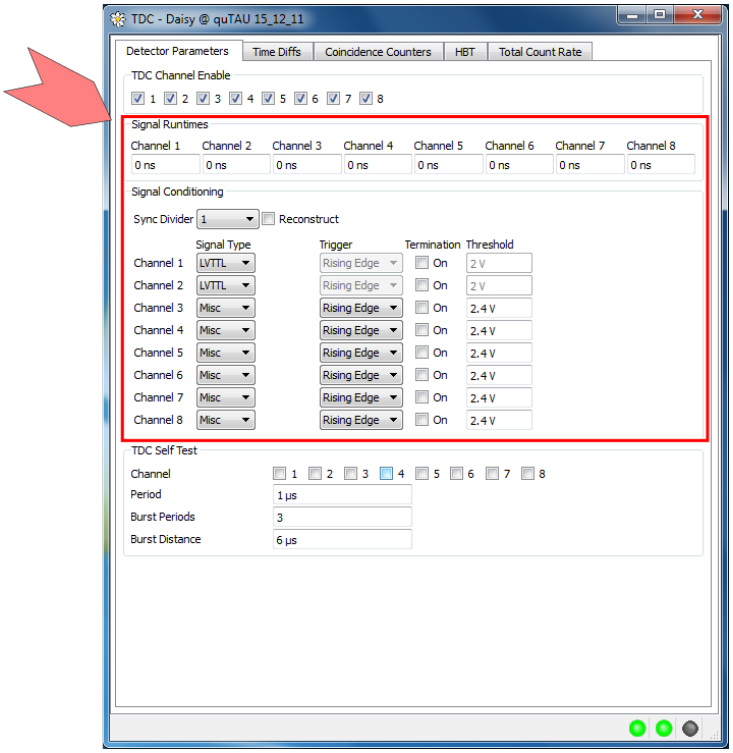

Figure 8: **Settings in he Daisy GUI.** The channel input settings can be changed via the controls in the highlighted area. Please note that appearance of this window may change depending on the used hardware; here, a quTAU (H) is used.

8.1.2 quTAU GUI

Simply click `Settings->Input Settings...` in the menu bar to display a dialog containing control elements for all available input settings, see Figure 9. Changes take immediate effect, you don't have to close the window.

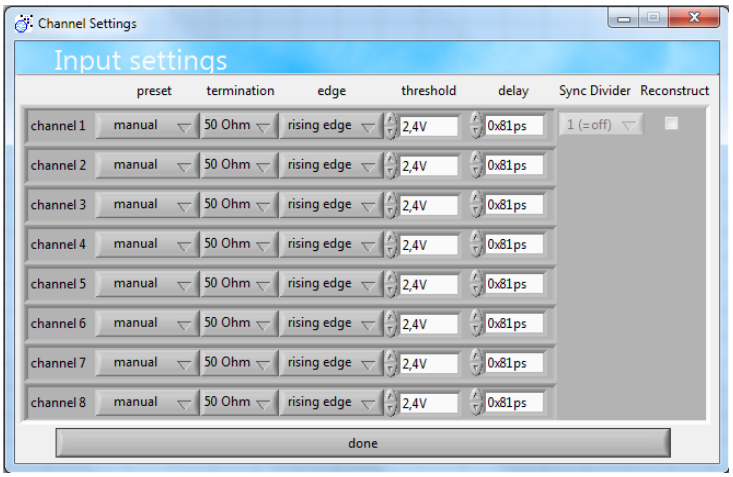

Figure 9: **The input settings dialog of the quTAU GUI.**

8.2 DLL Usage

Table 5: **Examples: Configure Input Parameter Settings**

LabView Example	`/userlib/labviewXX/examples/configure_IO.vi`
C Example	`/userlib/src/example[1,3,5].c`

A minimal working example is provided as `configure_IO.vi` (see figure 10).

Usage examples are also available in the `src` folder as `example[1,3,5].c`.

Functions relevant to the signal conditioning:
tdcbase.h:

TDC_DevType	**TDC_getDevType** ()
int	**TDC_configureSignalConditioning** (Int32 channel, TDC_SignalCond conditioning, BIn32 edge, BIn32 term, double threshold)
int	**TDC_getSignalConditioning** (Int32 channel, BIn32 *on, BIn32 *edge, BIn32 *term, double *threshold)
int	**TDC_configureSyncDivider** (Int32 divider, BIn32 reconstruct)
int	**TDC_getSyncDivider** (Int32 *divider, BIn32 *reconstruct)
int	**TDC_enableChannels** (Int32 channelMask)
int	**TDC_setChannelDelays** (const Int32 *delays)
int	**TDC_getChannelDelays** (Int32 *delays)
int	**TDC_switchTermination** (BIn32 on)
int	**TDC_enableTdcInput** (BIn32 enable)
int	**TDC_freezeBuffers** (BIn32 freeze)

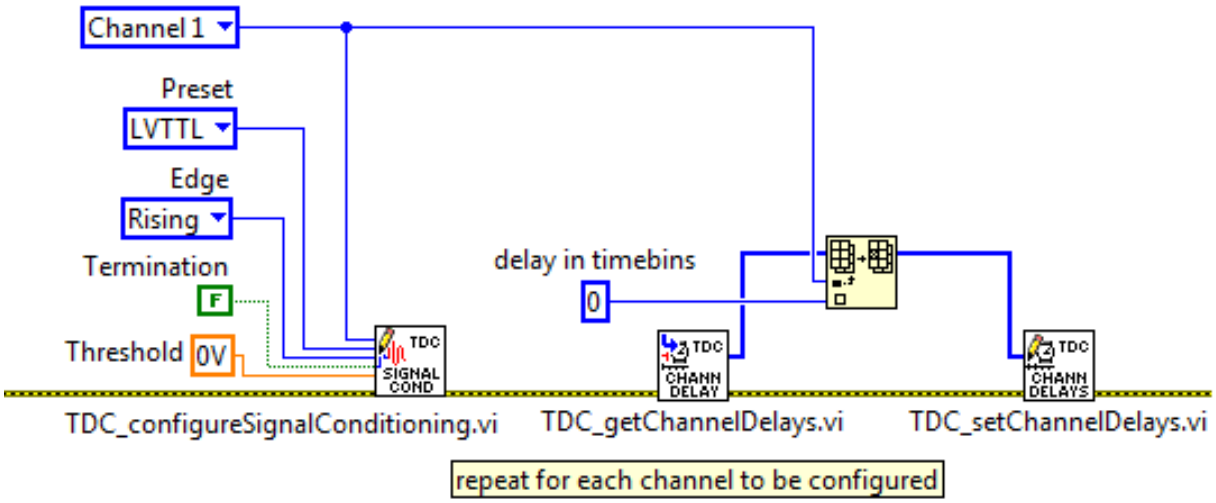

Figure 10: **Digital Input configuration,** see example `configure_IO.vi`. Please note that the logic level settings are set per channel and the channel delays as a set of eight.

9 (Coincidence) Counting

All single (each channel individually) and some coincidence rates (channel 1 to 4) are directly measured in the quTAU device, leading to 19 values (8 single count rates, 6 two-fold coincidences (1&2, 1&3, 1&4, 2&3, 2&4, 3&4), four three-fold coincidences (1&2&3, 1&2&4, 1&3&4, 2&3&4) and one four-fold coincidence (1&2&3&4)) being transferred to the PC by USB. Additionally to the input parameter settings, the user can specify the *integration time* and the *coincidence time window*.

9.1 Integration Time

The integration (or "exposure") time can be set to any ms value larger than 100 ms. All count rates will be shown according to this time window. This is also the minimum waiting time until a count rate value can be updated.

9.2 Coincidence Time Window

The coincidence time window can be set in integer multiples of the internal time unit ("Bins"). Two (or more) detection events will be counted as a coincident event if the difference of their time stamps is less or equal than the specified time window.

9.3 Realizations

Count rates can be monitored by both Daisy and the quTAU GUI. The command line interface can be used to output the last counter values.

9.3.1 Daisy

Count rate information can be accessed by switching to the Tab "Coincidence Counters". At the top, the required parameters can be set. All 19 current values are displayed in a bar plot. The plot updates once per integration time.

Additionally, the trend of a selected count rate can be displayed in a line graph.

9.3.2 quTAU GUI

A trend of all count rates is displayed in the Tab "Count Rates". On the right hand side, the required parameters can be set and individual count rates can be displayed or hidden using the check boxes.

If no device is connected, simulated data can be used to test and explore the interface. By right clicking on the graph, the displayed data can be exported.

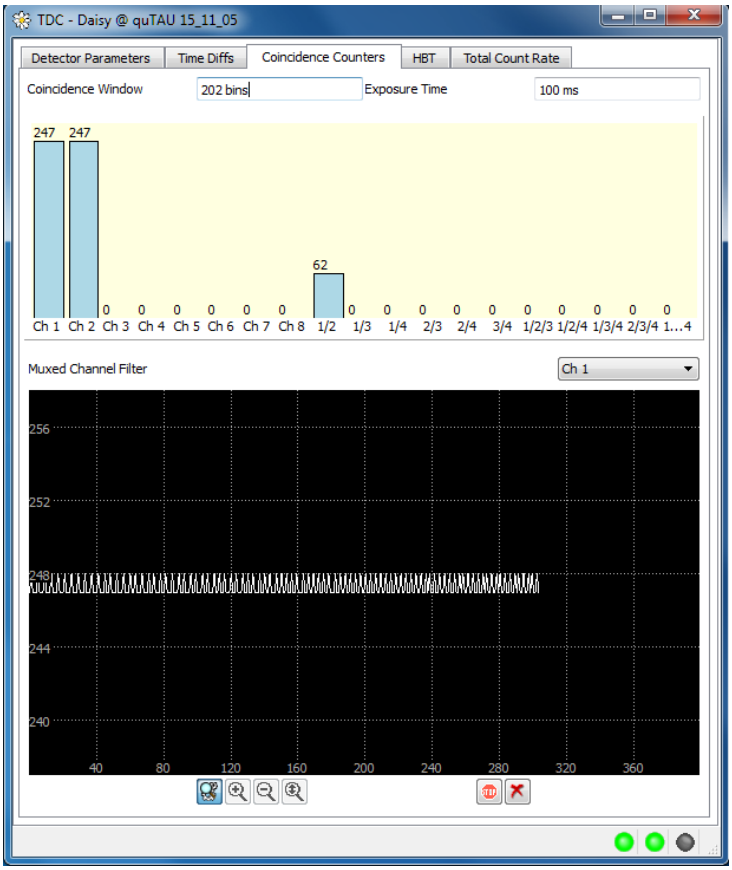

Figure 11: **Coincidence Counters Tab of the Daisy.** Parameters can be set at the top, data are displayed as a bar plot and additionally as a trend graph for a selected channel.

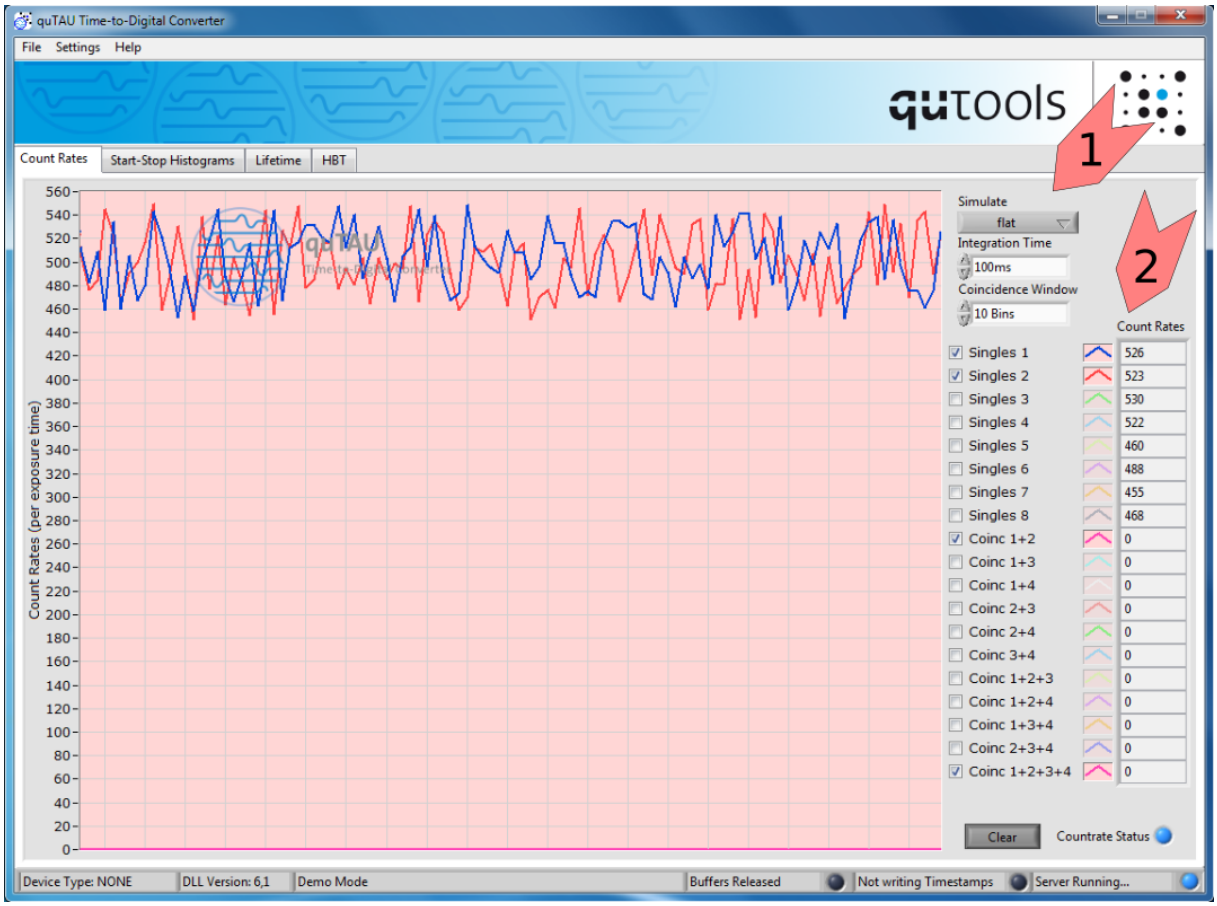

Figure 12: **Count Rates tab of the quTAU GUI.** Data are displayed in the graph on the left hand side, parameters can be set on the right hand side (1). Additionally, all count rates are also displayed numerically besides the plot legend (2).

9.3.3 Command Line Interface

The parameters **-x** <**exposure time in ms**> and **-W** <**coincidence window in Bins**> can be used to set exposure time and coincidence window. The last counter values are retrieved by the option **-C**.

9.4 DLL Usage

Table 6: **Examples: (Coincidence) Counting**

LabView Example	`/userlib/labviewXX/examples/countrate_plot.vi`
C Example	–

Once initialized, the single and coincidence rates can be retrieved from the device using `TDC_getCoincCounters` at any time. A minimal LabVIEW™ example (see Figure 13) is available as `countrate_plot.vi`. When polled periodically, the second argument `update` indicates the number of data updates since the last poll. The timebase for all rates is set by `TDC_setExposureTime` and the coincidence time window by `TDC_setCoincidenceWindow`.

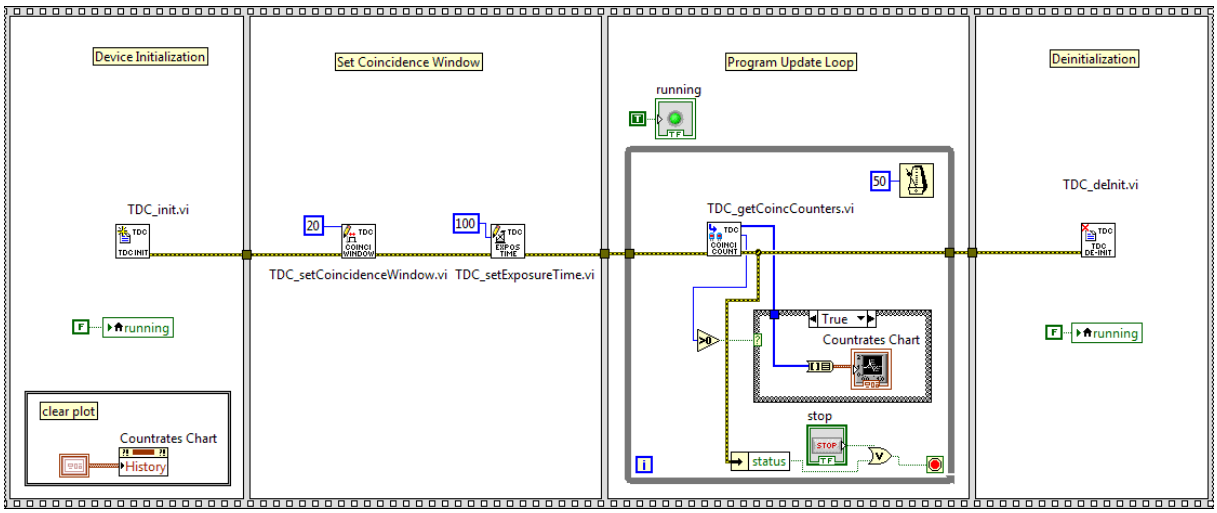

Figure 13: **Block diagram for a program to retrieve and display Single and Coincidence Rates,** see example `countrate_plot.vi`

Functions relevant to Single and Coincidence Rates:
tdcbase.h:

```
double   TDC_getTimebase ()
   int   TDC_setCoincidenceWindow (Int32 coincWin)
   int   TDC_setExposureTime (Int32 expTime)
   int   TDC_getDeviceParams (Int32 *channelMask, Int32 *coincWin, Int32 *expTime)
   int   TDC_getDataLost (BIn32 *lost)
   int   TDC_getCoincCounters (Int32 *data, Int32 *updates)
```

10 Time Stamping

The hardware and software allow you to use quTAU/quPSI devices to record time stamps of registered events.

10.1 Writing and loading Time Stamps

The time stamps can be saved using different file formats:

10.1.1 Text File (ASCII)

The time stamps will be written into a text file. There will be two columns:

1. The time stamp in base units, i.e. roughly 81 ps.
2. The channel (1..8).

10.1.2 Binary File 10 Bytes per Event

In this case, the time stamp of each event is written to the file in binary mode. The first 8 bytes correspond to the time stamp (in base units), while the following two bytes correspond to the channel number (also in binary). Since the 64 bits of timing information are more than are used internally, they will never overflow. (The internal time will overflow after roughly 67.6 days, see Section 4.2.)

10.1.3 Binary File 5 bytes per Event

Similar to 10.1.2, but with only 37 bits for the time stamp and 3 bits for the channel information. This overflows a lot sooner (roughly after 11 seconds). If there is no event for longer than 11 seconds, it will become impossible to reconstruct the time stamps correctly.

10.1.4 Digital Signature for Offline Analysis

If this option is enabled, the written files will contain a digital signature such that, using the supplied software, they can be read back in and analyzed offline, i.e. with the device unplugged.

10.1.5 Loading Time Stamps

The saved File can later be read back in for further analysis.

10.2 Software Realizations

Time stamps can be saved using the Daisy GUI, the quTAU GUI or the command line interface.

10.2.1 Daisy

The Daisy GUI is only capable of saving the time stamps in text format. You have to select the "Time Diffs" tab to start recording, see Figure 14

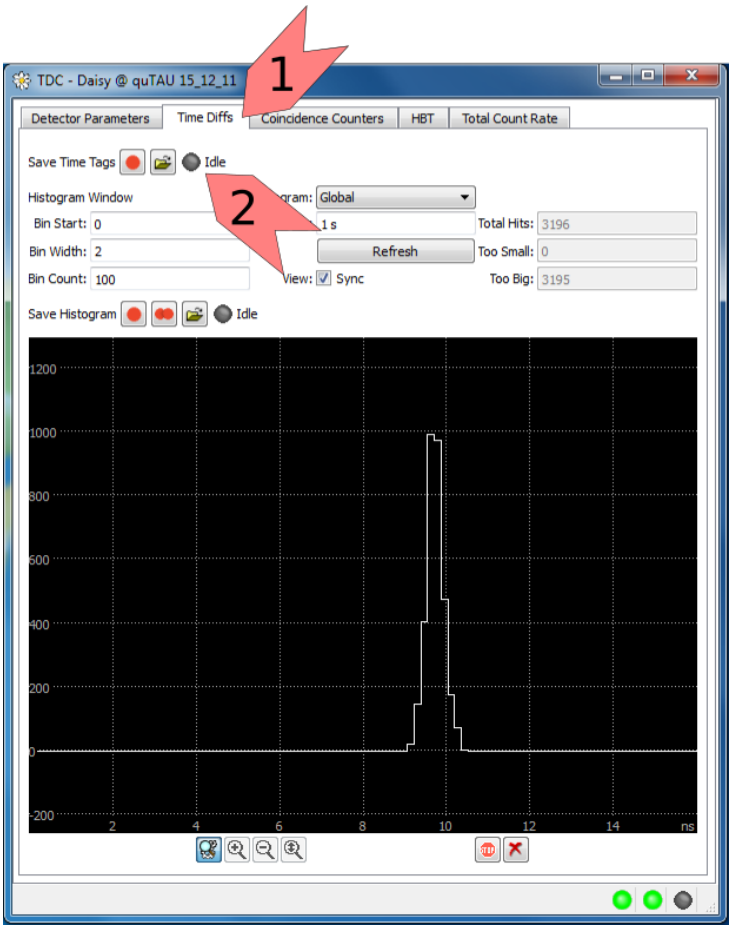

Figure 14: **Writing time stamps with the Daisy GUI.** Select the "Time Diffs" tab (1), then use the "'Save Time Tags"' area (2). Click on the folder symbol to select a file location and name and start recording by clicking the red "record" button. The LED will change color to green while writing. Another click on the "'record"' button stops the recording.

10.2.2 quTAU GUI

From the main window of the quTAU GUI, select `File -> Write Timestamps to File...` from the menu bar. A dialog will open (Figure 15), enabling you to specify the parameters

and start the recording. Here, it is possible to automatically stop recording after a specified time or after a certain file size is reached. During the recording, a status window appears,

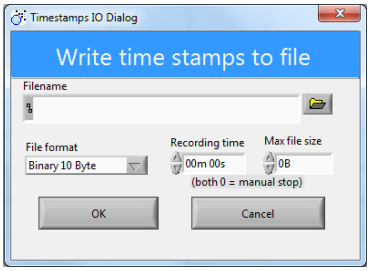

Figure 15: **The Write Timestamps Dialog.** You have to specify a file location and name by clicking on the folder symbol (1) and a file format (2). Additionally, you can set a maximum recording time (3) and/or a maximum file size (4). Click "Ok" to start recording time stamps or "Cancel" to abort.

displaying the time since the recording started and the file size of the save file, see Figure 16.

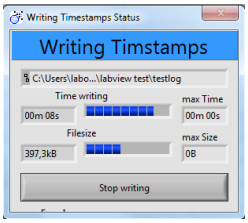

Figure 16: **The writing time stamps status window.** You can see the file name, the current recording time and file size and the chosen maximum time and file size. To stop recording time stamps, press the button.

Using the menu, you can also read time stamps from a file by clicking `File -> Read Timestamps from File...`.

10.2.3 Command Line Interface

The output format can be set to `ascii`, `binary`, `compressed` or `raw` using **-f** <**name**>. To save timetags, use **-F** <**filename**>.

10.3 DLL Usage

Table 7: **Examples: Writing Timestamps to File**

LabView Example	-
C Example	`/userlib/src/example4.c`

The different formats are encoded in the enum `TDC_FileFormat`. To enable recording, use the function `TDC_writeTimestamps` with the required format and a valid file name. The recording is terminated when `TDC_writeTimestamps` is called with an empty filename.

Alternatively, the timestamps can be retrieved as an array with an additional array containing the channel information. For this purpose, the time stamp buffer (set size with `TDC_setTimestampBufferSize`) is read using `TDC_getLastTimestamps`.

When loading saved timestamps for evaluation purposes with `TDC_readTimestamps` (from file) or `TDC_inputTimestamps` (from an array) it might be desirable to disable the device inputs with `TDC_enableTdcInput`.

Usage examples are also available in the `src` folder as `example4.c`.

Functions relevant for Time Stamp Handling:

tdcbase.h:

int **TDC_getDataLost** (BIn32 *lost)
int **TDC_setTimestampBufferSize** (Int32 size)
int **TDC_getTimestampBufferSize** (Int32 *size)
int **TDC_enableTdcInput** (BIn32 enable)
int **TDC_getLastTimestamps** (BIn32 reset, Int64 *timestamps, Int8 *channels, Int32 *valid)
int **TDC_writeTimestamps** (const char *filename, TDC_FileFormat format)
int **TDC_inputTimestamps** (const Int64 *timestamps, const Int8 *channels, Int32 count)
int **TDC_readTimestamps** (const char *filename, TDC_FileFormat format)

11 Histograms

All of our histograms are time difference histograms. They differ only in their usage of events as a start or stop signal.

11.1 Start-Stop Histograms

Start-Stop histograms use each detection event in one channel (e.g. channel A) as the start signal and only the first following detection event in another channel (e.g. channel B) as a stop signal. It is possible that A and B is the same channel, and, additionally, there is a channel independent Start-Stop histogram, where each event triggers a start or stop signal. In total, there are $8 \cdot 8 + 1 = 65$ Start-Stop histograms, with the 8 possibilities of channel A and B.

Example As an example, consider the events x_i, ordered by their timestamp, in the channels A and B depicted in Figure 17. The arrows show the considered time differences for the different histogram types.

For the A-A Start-Stop histogram, the time differences $x_2 - x_1$, $x_6 - x_2$, $x_8 - x_6$ and $x_9 - x_8$ would be recorded. In the A-B case, it would be $x_3 - x_2$, $x_7 - x_6$ and $x_{10} - x_9$. For the channel independent case, all $x_{n+1} - x_n$ result.

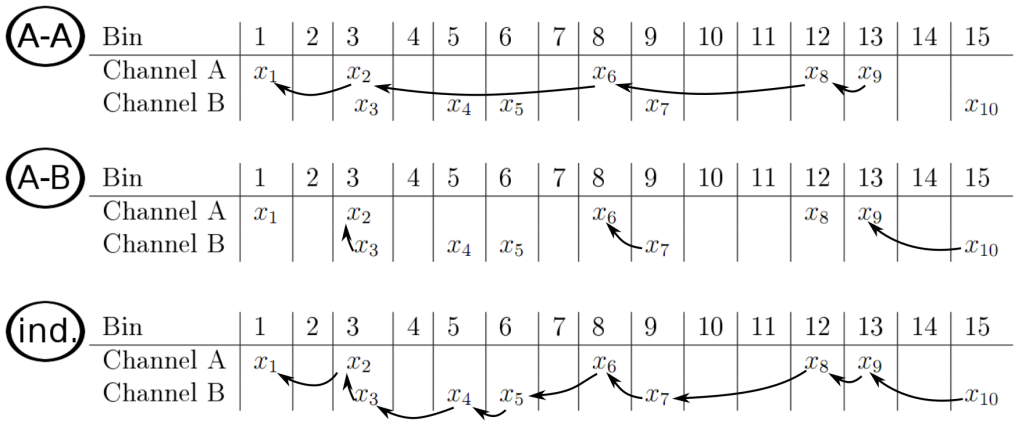

Figure 17: **Example of the determination of time differences for different start-stop histograms.** Three possibilities are shown: Start and stop event in the same channel (A-A), start and stop events in different channels (A-B) and channel independent (ind.). Each arrow starts on a STOP-event and points towards the respective START-event. Please note that, by convention, time differences of events happening in different channels but the same time bin are recorded as $x_A - x_B = 0$ for the (A-B) histogram with A<B.

11.1.1 Realization in the quTAU GUI

Start-Stop Histograms can be easily obtained by the quTAU GUI by switching to the "Start-Stop Histograms" tab, see Figure 18. Settings can be accessed on the right hand side, the retrieved data are shown in the plot on the left.

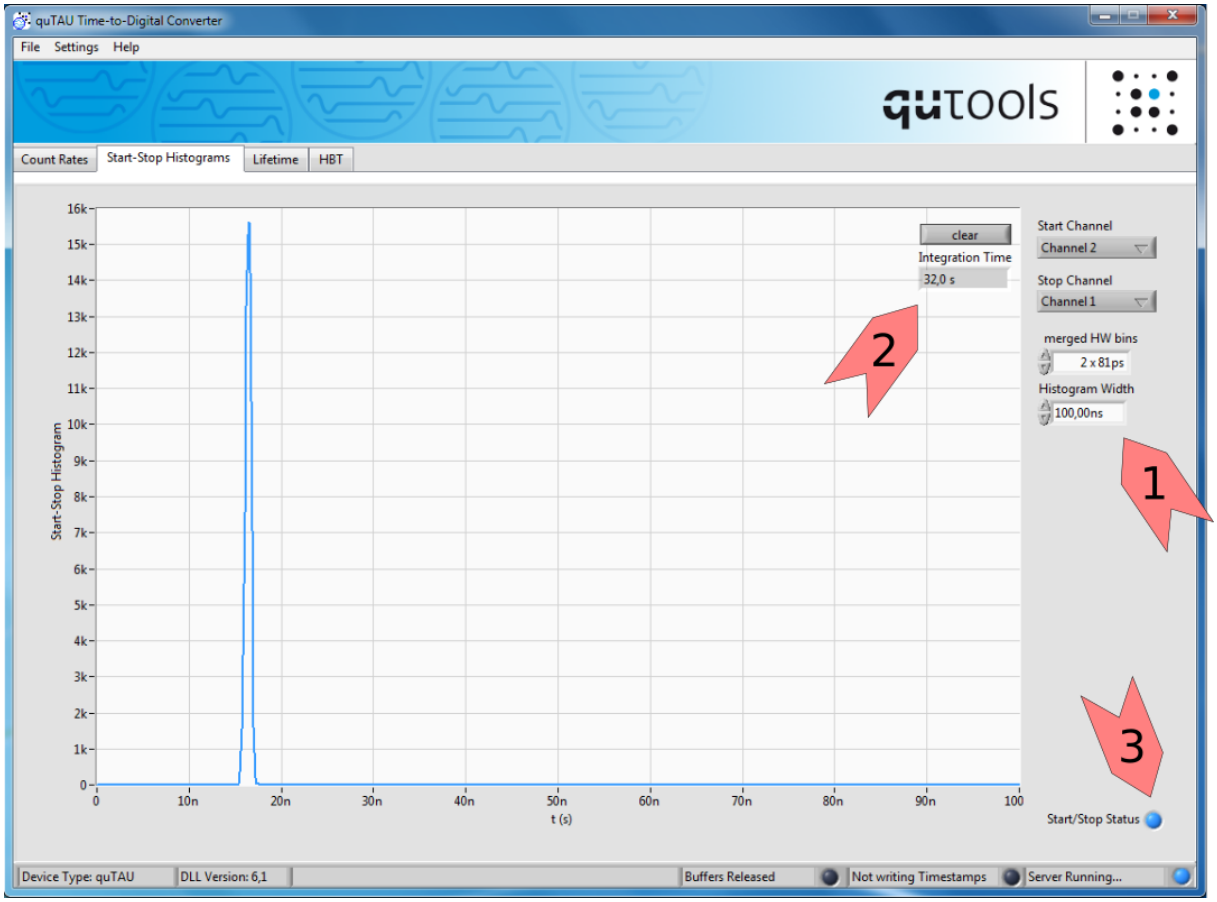

Figure 18: **Tab "Start-Stop Histograms" of the quTAU GUI.** On the left (1), Start and Stop channel can be specified. Additionally, you can change the number of merged HW bins (in order to circumvent the TDC chip problem mentioned in 4.1, choose an even number) and the histogram width. The total integration time (2) is also shown in the plot region. An LED (3) monitors the status of the operation.

11.1.2 DLL Usage

Table 8: **Examples: Start-Stop Histograms**

LabView Example	`/userlib/labviewXX/examples/start-stop.vi`
C Example	-

Example `start-stop.vi` describes how to retrieve start-stop histograms using the DLL (see fig. 19). The generation of start-stop histograms has to be enabled explicitly using `TDC_enableStartStop`. Further, the histogram width and resolution has to be specified using `TDC_setHistogramParams`. This is done using the two parameters "bincount" (the number of bars is the histogram) and "binwidth" (number of device time bins summarized in one histogram bin).

Once enabled, all available histograms are continuously generated. The parameters of `TDC_getHistogram` determine the channel combination to be retrieved. Additionally, an array with at least bincount entries has to be provided in the function call.

In case the histograms for multiple channel combinations are necessary, they can be retrieved one after the other. In order to get the histograms for the same integration time each, use `TDC_freezeBuffers`.

Functions relevant to Start Stop Histograms:
tdcstartstop.h:

int **TDC_enableStartStop** (BIn32 enable)
int **TDC_setHistogramParams** (Int32 binWidth, Int32 binCount)
int **TDC_getHistogramParams** (Int32 *binWidth, Int32 *binCount)
int **TDC_clearAllHistograms** ()
int **TDC_getHistogram** (Int32 chanA, Int32 chanB, BIn32 reset, Int32 *data, Int32 *count, Int32 *tooSmall, Int32 *tooLarge, Int32 *eventsA, Int32 *eventsB, Int64 *expTime)

11.2 Start-Multistop Histograms: Lifetime Measurements

Please note that this feature is only optional for quTAU devices with the hardware upgrade and part of the software extension quTAU (H+).

In contrast to the single-stop histograms discussed above, *all* STOP events (not only the first one) following a given START event are used to calculate time differences and make up the histogram. Therefore, this method is perfect for lifetime measurements.

11.2.1 Model functions

The most commonly used models for lifetime measurements are *exponential decay*, *double exponential decay*, and the *Kohlrausch decay function*.

Standard: Exponential Decay

$$f_1(t) = I_0 \cdot \mathrm{e}^{-t/\tau_0} \tag{1}$$

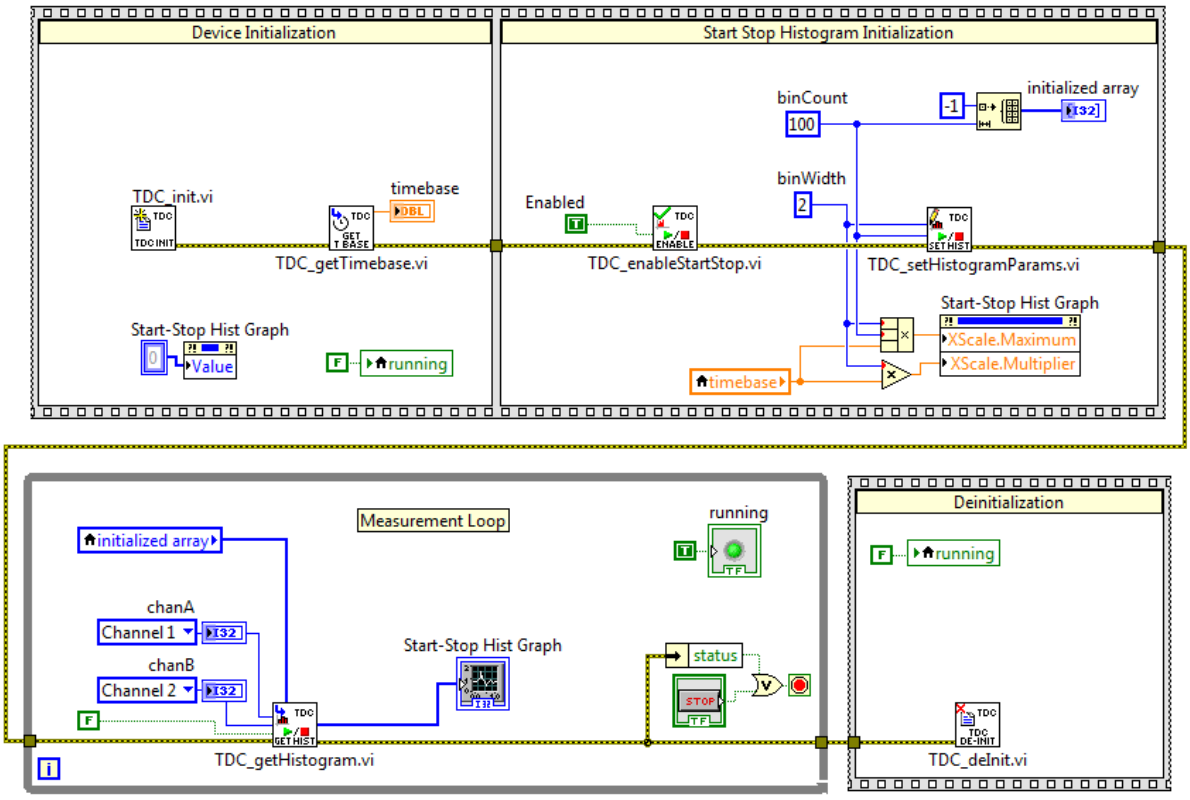

Figure 19: **Block diagram for start-stop histograms,** see example `start-stop.vi`.

Double Exponential (FLIM-FRET)

$$f_2(t) = I_0 \left(\alpha \cdot \mathrm{e}^{-t/\tau_0} + (1-\alpha) \cdot \mathrm{e}^{-t/\tau_1}\right) \tag{2}$$

Kohlrausch Decay Function, Stretched Exponential

$$f_3(t) = I_0 \cdot \mathrm{e}^{-(t/\tau_0)^\beta} \tag{3}$$

11.2.2 Realization in the quTAU GUI

Lifetime Measurements can be obtained and analyzed in the quTAU GUI, see Figure 20. Additionally, the data can be fitted to one of the models explained above. For testing purposes or if no device is connected, data can also be simulated (and analyzed normally). Settings can be accessed on the bottom, the retrieved data are shown in the plot at the top.

11.2.3 DLL Usage

Table 9: **Examples: Lifetime Measurements**

LabView Example	`/userlib/labviewXX/examples/lifetime.vi`
C Example	`/userlib/src/example5.c`

The quTAU software assists in lifetime measurements. The work flow can be seen in the LabVIEW™ example `lifetime.vi` (see Figure 21). The lifetime feature is optional, use `TDC_checkFeatureLifeTime` to check for its availability.

Again, the lifetime feature has to be enabled explicitly after the histogram parameters have been set. In this case, the number of bins ("bincount") is for the positive bins only. Additionally, a storage container for the measurement (and a second one for the fit if needed) has to be generated. This is done with the function call `TDC_createLftFunction`.

For the procedure to get the histograms from the device and to fit one of the provided model functions confer Figure 21.

Note, that the size of the containers/data sets has to match the number of bins in the histogram. Therefore, if you change the histogram size you need to release and recreate the data sets with `TDC_releaseLftFunction` and `TDC_createLftFunction`.

Usage examples are also available in the `src` folder as `example[5].c`.

Functions relevant to Lifetime Measurements:
tdcbase.h:

BIn32 **TDC_checkFeatureLifeTime** ()

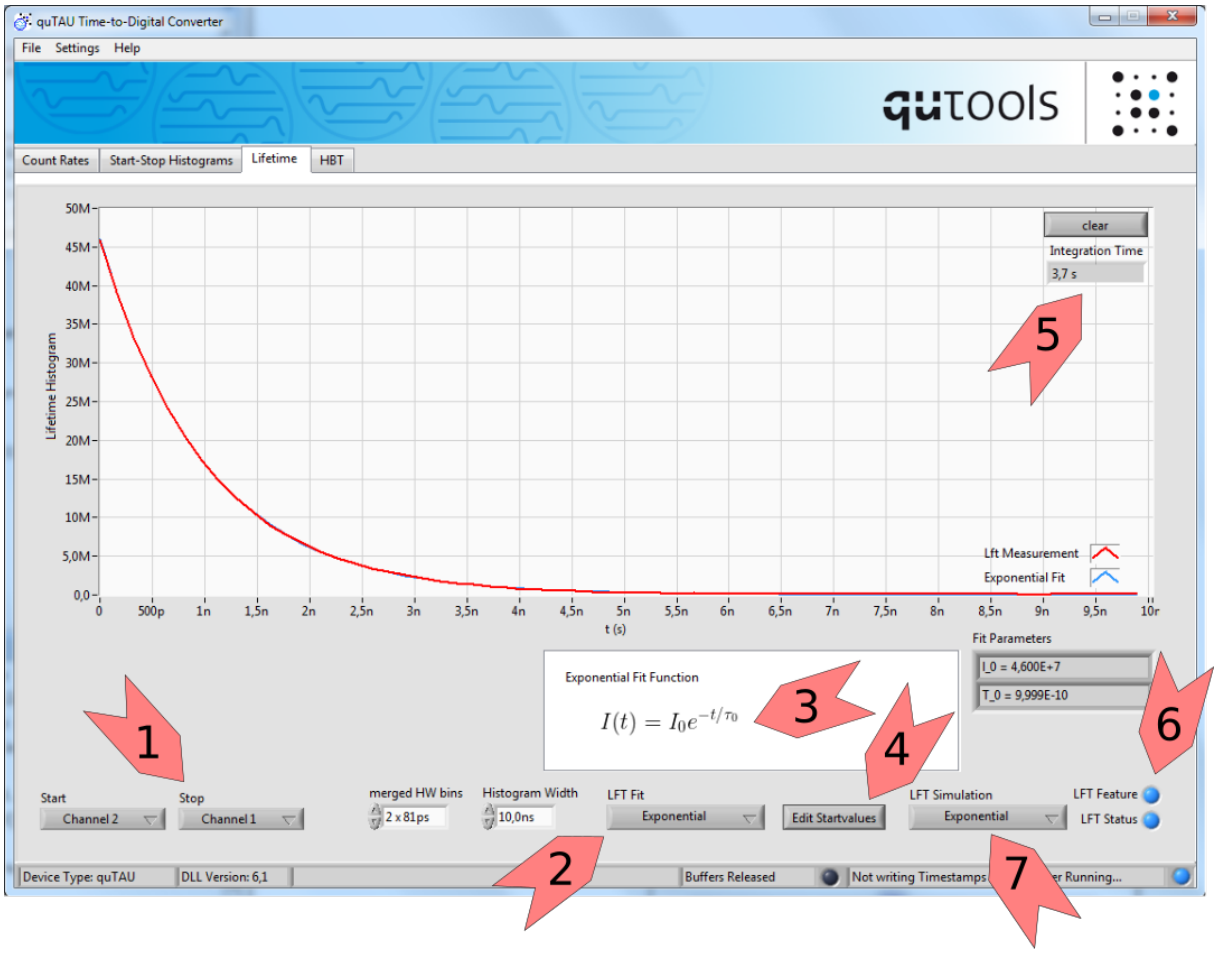

Figure 20: **Tab "Lifetime" of the quTAU GUI.** On the bottom (1), Start and Stop channel can be specified. Additionally, you can change the number of merged HW bins (in order to circumvent the TDC chip problem mentioned in 4.1) and the histogram width. In order to analyze the obtained data, a fit to one of the three models described above can be chosen (2). The fit function will be displayed (3). Edit the start values (4) if the fit does not work properly. The total integration time (5) is also shown in the plot region. LEDs (6) monitors the status of the operation. For testing purposes, a simulation of one of the models can be started (7).

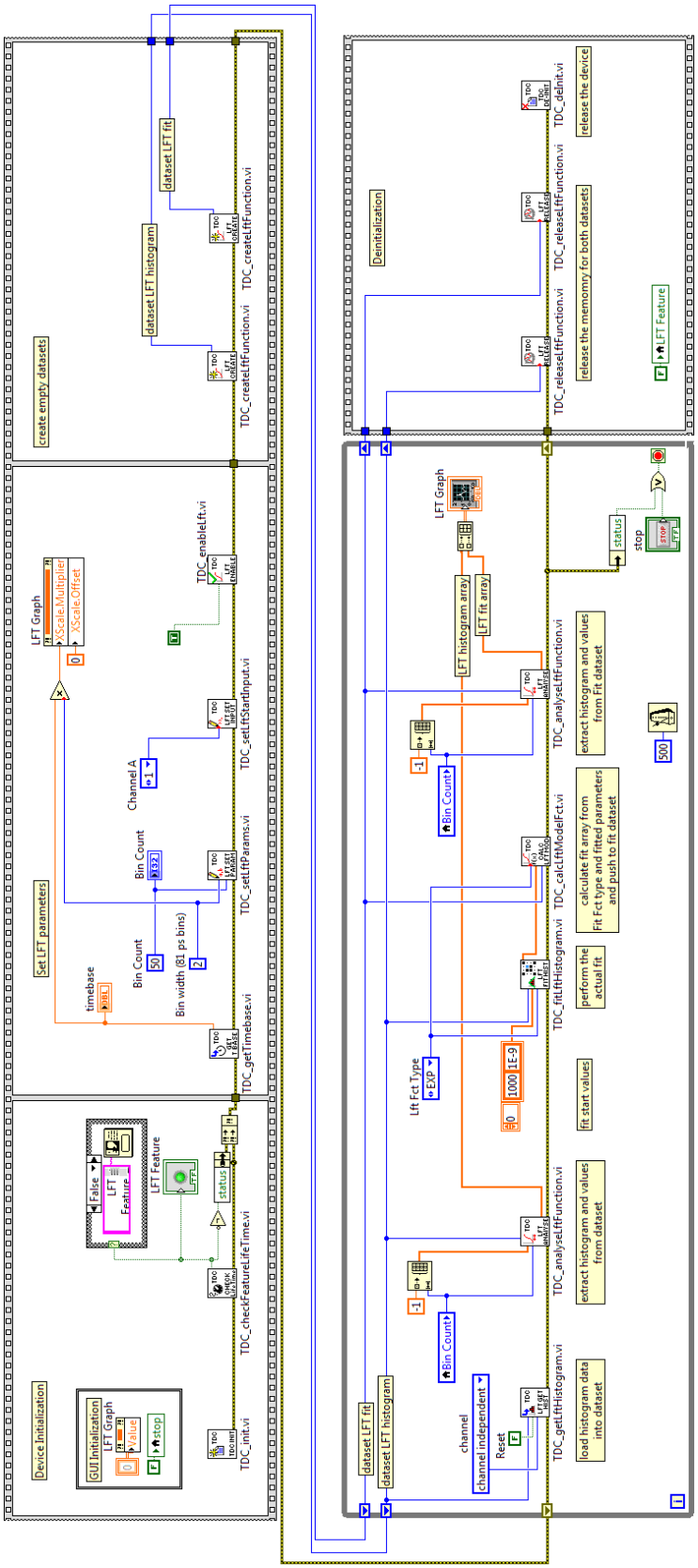

Figure 21: **Lifetime measurements,** see example `lifetime.vi`. The `TDC_analyzeLftFunction` function is a convenient function to retrieve single values and the histogram array from the data set.

tdclifetime.h:

int	**TDC_enableLft** (BIn32 enable)
int	**TDC_setLftParams** (Int32 binWidth, Int32 binCount)
int	**TDC_getLftParams** (Int32 *binWidth, Int32 *binCount)
int	**TDC_setLftStartInput** (Int32 startChan)
int	**TDC_getLftStartInput** (Int32 *startChan)
int	**TDC_resetLftHistograms** (void)
TDC_LftFunction *	**TDC_createLftFunction** (void)
void	**TDC_releaseLftFunction** (TDC_LftFunction *fct)
void	**TDC_analyseLftFunction** (const TDC_LftFunction *fct, Int32 *capacity, Int32 *size, Int32 *binWidth, double *values, Int32 bufSize)
int	**TDC_getLftHistogram** (Int32 channel, BIn32 reset, TDC_LftFunction *fct, Int32 *tooBig, Int32 *startEvts, Int32 *stopEvts, Int64 *expTime)
int	**TDC_calcLftModelFct** (LFT_FctType fctType, const double *params, TDC_LftFunction *fct)
int	**TDC_generateLftDemo** (LFT_FctType fctType, const double *params, double noiseLv)
int	**TDC_fitLftHistogram** (const TDC_LftFunction *fct, LFT_FctType fitType, const double *startParams, double *fitParams, Int32 *iterations)

11.3 Correlation Measurements: HBT

To analyze some light source, one often uses the second order correlation function,

$$g^{(2)}\tau = \langle I(t) \cdot I(t+\tau) \rangle \, / \, \langle I(t) \rangle^2 \, . \tag{4}$$

It basically tells you when to expect the next photon when there has just been one here. A value of one corresponds to no dependence on the previous detection time. A value larger than 1 means that there is some positive correlation (two photons are more likely to be detected within the given time difference), while a value smaller than 1 tells you that you are less likely to see these time differences (antibunching).

To measure the second order correlation function, cross-correlation histograms are employed. Neither start-stop, nor start-multistop histograms are the methods of choice, although especially start-stop measurements are often used. The difference will be briefly discussed here.

11.3.1 Hanbury-Brown and Twiss setup

The name of these kind of measurements stems from the setup devised by Hanbury-Brown and Twiss shown in Figure 22. It consists of two single photon detectors located at each output of a 50/50 beam splitter. Time difference measurements of photon detections can then be used to analyze the light sent into the input of the beam splitter.

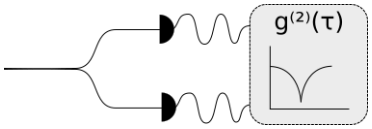

Figure 22: **Hanbury-Brown and Twiss setup.**

11.3.2 $g^{(2)}$ function and start-stop histograms of coherent light

Photons in coherent light (e.g. that of a laser) are independent of each other, such that there is a constant probability p to detect a photon in each time interval. This leads to Poissonian statistics when counting the number of photons and also to a constant second order correlation function,

$$g^{(2)}(\tau) = 1, \tag{5}$$

where p is normalized to 1.

In contrast, when a start-stop histogram is recorded, you will see a different picture. Because all but one STOP-event between to START-pulses are ignored, you have the probability $p(1-p)^{n-1}$ to capture a STOP-event in the nth bin after a START event in bin 0. Thus, the correlation function will exhibit an exponential decrease instead of a constant. Furthermore, if the used detectors have a dead time τ_D, i.e. they can't register a second photon within a certain time after the first, this exponential decrease will start only after τ_D of the respective detector.

Depending on the time scale of the phenomenon to be observed, this may or may not be a problem for your measurement. Therefore, we always advise to use cross-correlation histograms to measure the $g^{(2)}$ function.

11.3.3 Model functions

The most common models for the $g^{(2)}$ correlation function are shown below. With these, *thermal* photons, *antibunching*, and *three-level-systems* can be described.

Thermal

$$g^{(2)}_{th}(\tau) = A \cdot \mathrm{e}^{-\frac{(\tau-\tau_0)^2}{2c^2}} + B \tag{6}$$

Antibunching

$$g^{(2)}_{ab}(\tau) = 1 - \mathrm{e}^{-\frac{|\tau-\tau_0|}{\tau_a}} \tag{7}$$

Three-Level-System

$$g^{(2)}_{tl}(\tau) = 1 + p_f^2 \left(c \cdot \mathrm{e}^{-\frac{|\tau-\tau_0|}{\tau_b}} - (1+c) \cdot \mathrm{e}^{-\frac{|\tau-\tau_0|}{\tau_a}} \right) \tag{8}$$

Normalization Please note that the measured histogram data are normalized with a factor

$$N = R_1 R_2 T W, \tag{9}$$

with countrates R_1 and R_2, total integration time T and histogram bin width W.

11.3.4 Realization in the quTAU GUI

HBT Measurements can be obtained and analyzed in the quTAU GUI, see Figure 23. For testing purposes or if no device is connected, data can also be simulated (and analyzed normally). Settings can be accessed on the bottom, the retrieved data are shown in the plot at the top.

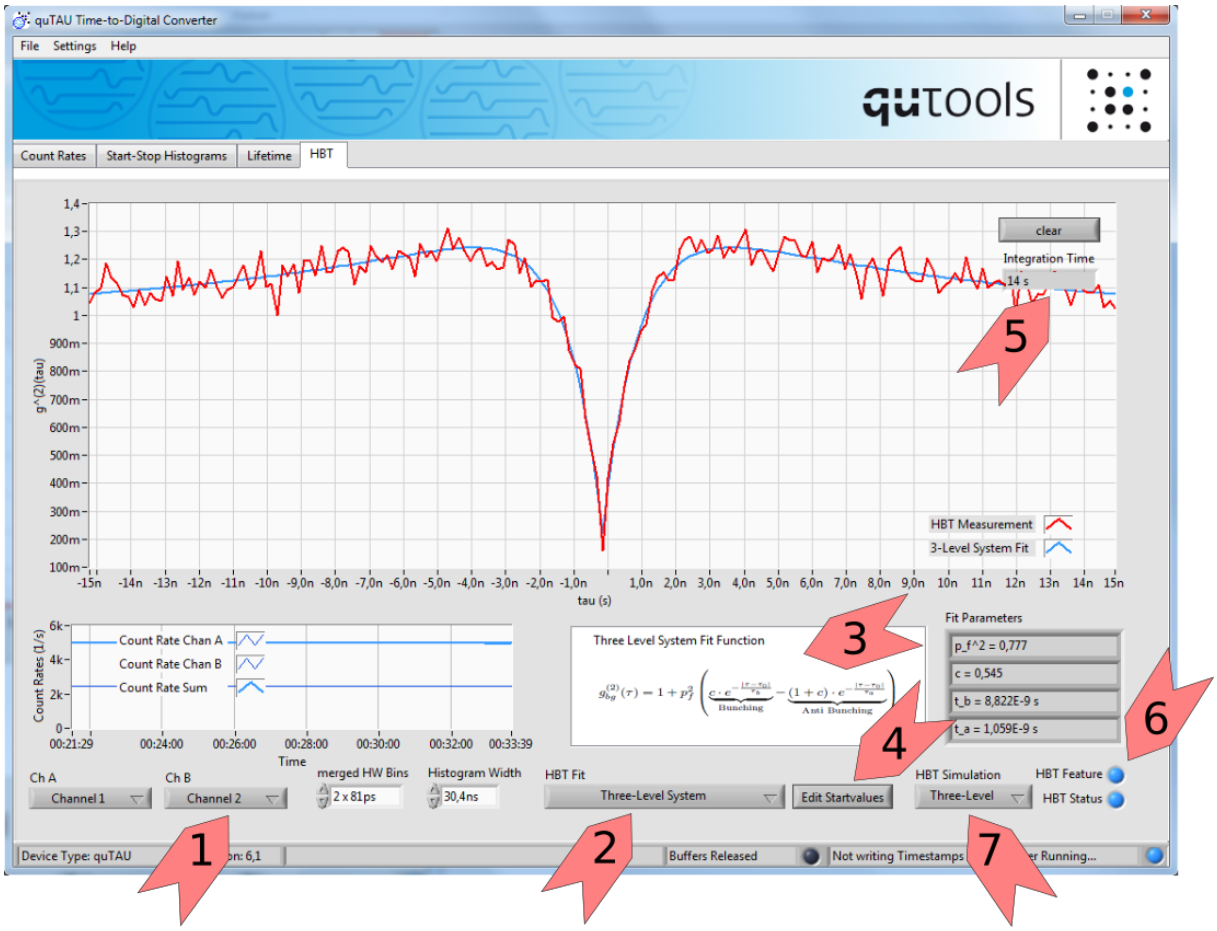

Figure 23: **Tab "HBT" of the quTAU GUI.** On the bottom (1), Start and Stop channel can be specified. Additionally, you can change the number of merged HW bins (in order to circumvent the TDC chip problem mentioned in 4.1) and the histogram width. In order to analyze the obtained data, a fit to one of the three models described above (with the option to include detector jitter and an offset) can be chosen (2). The fit function and the obtained parameters will be displayed (3). Edit the start values (4) if the fit does not work properly. The total integration time (5) is also shown in the plot region. LEDs (6) monitor the status of the operation. For testing purposes, a simulation of one of the models can be started (7).

11.3.5 DLL Usage

Table 10: **Examples: HBT Measurements**

LabView Example	`/userlib/labviewXX/examples/hbt_measurement.vi`
C Example	`/userlib/src/example2.c`

The quTAU software assists in HBT measurements. The work flow can be seen in the LabVIEW™ example `hbt_measurement.vi` (see Figure 24). The HBT feature is optional, use `TDC_checkFeatureHbt` to check for its availability.

Again, the HBT feature has to be enabled explicitly after the histogram parameters have been set. As for the Lifetime feature, the number of bins ("bincount") is for the positive bins only and a storage container for the measurement (and a second one for the fit if needed) has to be generated. This is done with the function call `TDC_createHbtFunction`.

For the procedure to get the histograms from the device and to fit one of the provided model functions confer Figure 24.

Note, that the size of the containers/data sets has to match the number of bins in the histogram. Therefore, if you change the histogram size you need to release and recreate the data sets with `TDC_releaseHbtFunction` and `TDC_createHbtFunction`.

Usage examples are also available in the `src` folder as `example[2].c`.

Functions relevant to HBT Measurements:

tdcbase.h:

BIn32	**TDC_checkFeatureHbt** ()

tdchbt.h:

int	**TDC_enableHbt** (BIn32 enable)
int	**TDC_setHbtParams** (Int32 binWidth, Int32 binCount)
int	**TDC_getHbtParams** (Int32 *binWidth, Int32 *binCount)
int	**TDC_setHbtDetectorParams** (double jitter)
int	**TDC_getHbtDetectorParams** (double *jitter)
int	**TDC_setHbtInput** (Int32 channel1, Int32 channel2)
int	**TDC_getHbtInput** (Int32 *channel1, Int32 *channel2)
int	**TDC_switchHbtInternalApds** (BIn32 internal)
int	**TDC_resetHbtCorrelations** ()
int	**TDC_getHbtEventCount** (Int64 *totalCount, Int64 *lastCount, double *lastRate)
int	**TDC_getHbtIntegrationTime** (double *intTime)
int	**TDC_getHbtCorrelations** (BIn32 forward, TDC_HbtFunction *fct)
int	**TDC_calcHbtG2** (TDC_HbtFunction *fct)
int	**TDC_fitHbtG2** (const TDC_HbtFunction *fct, HBT_FctType fitType, const double *startParams, double *fitParams, Int32 *iterations)
const double *	**TDC_getHbtFitStartParams** (HBT_FctType fctType)
int	**TDC_calcHbtModelFct** (HBT_FctType fctType, const double *params, TDC_HbtFunction *fct)
int	**TDC_generateHbtDemo** (HBT_FctType fctType, const double *params, double noiseLv)
TDC_HbtFunction *	**TDC_createHbtFunction** (void)
void	**TDC_releaseHbtFunction** (TDC_HbtFunction *fct)
void	**TDC_analyseHbtFunction** (const TDC_HbtFunction *fct, Int32 *capacity, Int32 *size, Int32 *binWidth, Int32 *iOffset, double *values, Int32 bufSize)

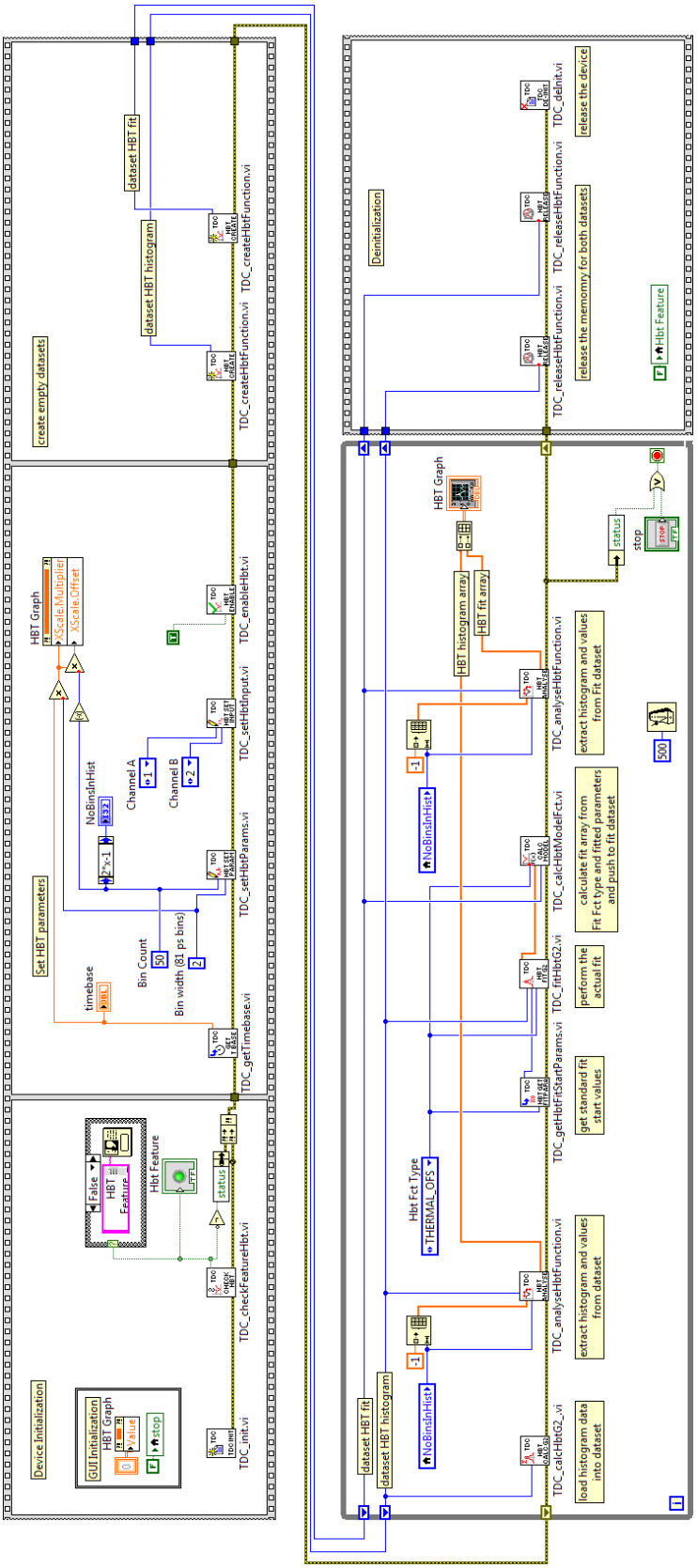

Figure 24: **HBT measurements,** see example `hbt_measurement.vi`. The `TDC_analyzeHbtFunction` function is a convenient function to retrieve single values and the histogram array from the data set.

12 Simulation and Testing: Demo Mode

For simulation and testing purposes, a demo mode is included. It is automatically accessed by the quTAU GUI when no suitable device is found. For countrate simulation, the simulation happens only in the GUI software, but for lifetime and HBT measurements, the demo mode is also included in the DLL to test the respective analyzing functions of the DLL.

12.1 quTAU GUI Simulation

You can find an option to simulate in the countrate, lifetime, and HBT panels. Events are generated to fit the specified model. Please note that it is not possible (yet) to generate demo timestamps for Lifetime and HBT measurements simultaneously; if one demo is started, the other is paused. Nevertheless, it is of course possible to run both panels simultaneously using a real device.

12.2 DLL Usage

For simulation and testing, events can be generated by the software as if they were produced by real signals. There are several options:

generate timestamps: Using the function `TDC_generateTimestamps` the buffers for all histograms (start-stop, lifetime, HBT) are fed the requested number of timestamps once.

generate demo: For the histogram types lifetime and HBT separate simulation functions exist. These generate time differences with the specified characteristics (exponential decay, anti bunching, etc.). In contrast to generate timestamps above, these functions enable or disable a continuous stream of simulated data.

Usage examples are also available in the `src` folder as `example[0,2,4].c`.

Functions relevant to Data Simulation

tdcbase.h:

int **TDC_generateTimestamps** (TDC_SimType type, double *par, Int32 count)

tdchbt.h:

int **TDC_generateHbtDemo** (HBT_FctType fctType, const double *params, double noiseLv)

tdclifetime.h:

int **TDC_generateLftDemo** (LFT_FctType fctType, const double *params, double noiseLv)

13 Differences using a quPSI

The same software realizations and user library can be used with a quPSI device. Just be sure to take note of the following differences.

13.1 quTAU GUI

When using the quTAU GUI with a quPSI device, please note that external countrates are detected in channel 1 (Sync), 3 and 4, while the internal APDs countrates are displayed in channel 5 and 6. There is also a fixed sync divider value of 1024 for channel 1. In the settings window, only the first 3 rows are used to change input settings for the external sources.

In the HBT tab of the GUI, there is now the option to switch between internal and external APDs for the analysis (instead of being able to freely choose channels).

13.2 DLL Usage

Please note that the function `TDC_setHbtInput` used to set the channels for HBT-analysis will return an error when used with a quPSI device. Instead, use `TDC_switchHbtInternalApds` to switch between internal and external APD usage. Other than that, functions will behave similarly for the quTAU and quPSI devices.