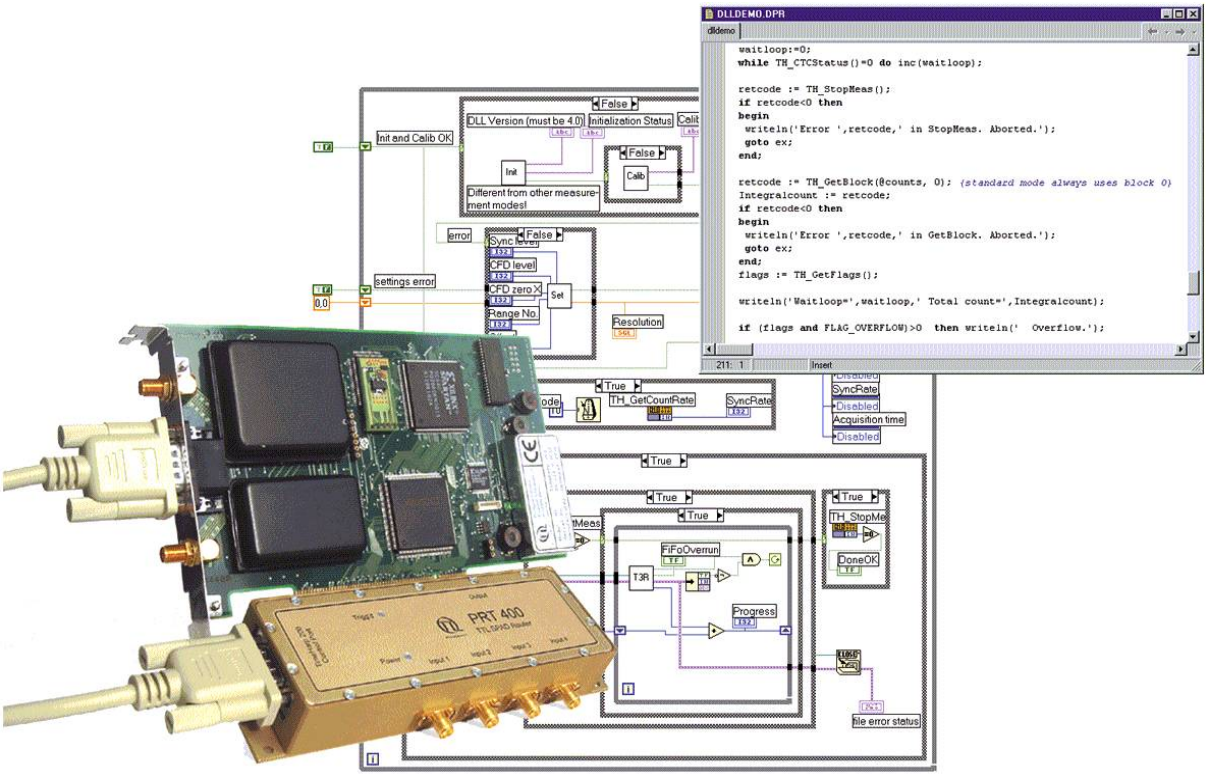
TimeHarp 200

Time Measurement Histogram
Accumulating Real-time Processor

PicoQuant GmbH
Unternehmen für optoelektronische
Forschung und Entwicklung

THLib.DLL Programming Library
for Custom Application Development



User's Manual

Version 6.1 - February 2009

Table of Contents

1. Introduction

The TimeHarp 200 is a compact easy-to-use TCSPC system on a single PCI board. It includes all components traditionally contained in bulky racks. Its integrated lean design keeps cost down, improves reliability and simplifies calibration. The circuit allows high measurement rates up to 3 Mcounts/s and provides a time resolution of less than 40 ps. The input triggers (detector and sync) are programmable for a wide range of input signals. The detector input has a programmable Constant Fraction Discriminator (CFD). These specifications qualify the TimeHarp 200 for use with all common single photon detectors such as Single Photon Avalanche Diodes (SPADs) and Photomultiplier Tube (PMT) modules (PMT via preamp). The time resolution is well matched to these detectors and the overall Instrument Response Function (IRF) will usually not be dominated by the TimeHarp electronics. Similarly inexpensive and easy-to-use diode lasers such as the PDL 800-B with interchangeable laser heads can be used as an excitation source well matched to the time resolution offered by the detector and the electronics. Overall IRF widths of 300 ps FWHM can be achieved with inexpensive PMTs and diode lasers. This permits lifetime measurements down to some tens of picoseconds with deconvolution e.g. via the FluoFit multiexponential Fluorescence Decay Fit Software. For more information on the TimeHarp 200 hardware and software please consult the TimeHarp 200 manual. For details on the method of Time-Correlated Single Photon Counting, please refer to our TechNote on TCSPC.

The TimeHarp 200 standard software provides functions such as the setting of measurement parameters, display of results, loading and saving of measurement parameters and histogram curves. Important measurement characteristics such as count rate, count maximum and position, histogram width (FWHM) are displayed continuously. While these features will meet many of the routine demands, advanced users may want to include the TimeHarp's functionality in their own automated measurement systems with their own software. In particular where the measurement must be interlinked or synchronized with other processes this approach may be of interest. For this purpose a programming library is provided as a Dynamic Link Library (DLL) for Windows. The programming library 'THLib.dll' supports custom programming in all major 32 bit programming languages, notably C, C++, C#, Delphi, Visual Basic, MATLAB and LabVIEW. This manual describes the installation and use of the TimeHarp programming library THLib.dll and explains the associated demo programs. Please read both this manual and the TimeHarp manual before beginning your own software development with the DLL. The TimeHarp is a sophisticated real-time measurement system. In order to work with the board using the TimeHarp DLL, sound knowledge in your chosen programming language is required.

There is also a programming library for Linux which is not covered here. Please check your distribution media and/or request additional information if you are interested in using this platform.

2. General Notes

This version of the TimeHarp 200 programming library is suitable for Windows 2000, XP and Vista. The x64 editions of Windows XP and Vista are also supported, although only for use by 32 bit applications. The new version 6.1 provides a new device driver for the TimeHarp board and new demos for C# and MATLAB. Apart from this, there is no functional change compared to v6.0. Applications should work unchanged, except a possible need for adaption of the version check.

The Library has been tested with MinGW 2.0 (free compiler for Windows, 32 bit), MSVC++ 6.0, Borland C++ 5.5, Visual C# 2005, Mono 2.0, Delphi 6.0, Lazarus 0.9.24, MATLAB 7.3, LabView 8.0 and Visual Basic 6.0.

This manual assumes that you have read the TimeHarp 200 manual that came with your TimeHarp board and that you have experience with the chosen programming language. References to the TimeHarp manual will be made where necessary.

The library supports both histogramming and TTTR modes including multi-channel measurements (routing) with PRT/NRT 400 and up to 4 detectors.

Users who purchased any older version of the library will receive free updates when they are available. For this purpose, please register by sending email to *info@picoquant.com* with your name, your TimeHarp 200 serial number and the email address you wish to have the updates sent to.

Users upgrading from earlier versions of the TimeHarp 200 DLL may need to adapt their programs. Some changes are usually necessary to accommodate new measurement modes and improvements. However, the required changes are usually minimal and will be explained in the manual (especially check the notes marked in red in section 9).

Note that despite of our efforts to keep changes minimal, data structures, program flow and function calls may still change in future versions without advance notice. Users must maintain appropriate version checking in order to avoid troble with incompatibilities. There is a function call tat you can use to retrieve the version number (see section 9). Note that this call returns only the major two digits of the version (e.g. 6.1). The DLL actually has two further sub-version digits, so that the complete version number has four digits (e.g. 6.1.0.0). They are shown only in the Windows file properties. These sub-digits help to identify intermediate versions that may have been released for bug fixes. The interface of releases with identical major versions will be the same.

Disclaimer

PicoQuant GmbH disclaims all warranties with regard to this software including all implied warranties of merchantability and fitness. In no case shall PicoQuant GmbH be liable for any direct, indirect or consequential damages or any material or immaterial damages whatsoever resulting from loss of data, time or profits arising from use or performance of this software. Demo code is provided ‘as is’ without any warranties as to fitness for any purpose.

License and Copyright Notice

With this product you have purchased a license to use the 200 programming library on a single PC. You have not purchased any other rights to the software itself. The software is protected by copyright and intellectual property laws. You may not distribute the software to third parties or reverse engineer, decompile or disassemble the software or part thereof. You may use and modify demo code to create your own software. Original or modified demo code may be re-distributed, provided that the original disclaimer and copyright notes are not removed from it.

TimeHarp is a registered trademark of PicoQuant GmbH.

Other trademarks, notably those related to operating systems, programming languages and their components etc. are property of their respective owners and are used here for explanatory purposes only, to the owner's benefit, without the intent to infringe.

3. Firmware Update

The TimeHarp 200 programming library requires a firmware update of your TimeHarp 200 board, unless you already bought it with the DLL option installed. The update is performed by DLLUPD.EXE. The update only needs to be done once. If necessary, perform the following steps to install the update:

(see your TimeHarp manual for steps 1..3 listed below)

1. Make sure your TimeHarp 200 board is installed correctly in the PC.
2. Check that the standard TimeHarp 200 software runs correctly.
3. Make sure to exit the TimeHarp 200 software.
4. Start the program DLLUPD.EXE directly from the floppy disk or from a temporary disk drive.
5. Follow the instructions. Do not interrupt the actual update progress, it may take a minute or so. The program will report successful completion.

You are then ready to use the TimeHarp DLL. See the sections below for hints how to install and use it. See the subfolder DEMO in your THLib installation folder for sample code.

4. Installation of the DLL

The TimeHarp 200 programming library will not be installed by the standard TimeHarp 200 software setup. The standard "interactive" TimeHarp 200 data acquisition software does not require the DLL, which is provided for custom application programming only. Vice versa, your custom program will only require the DLL and drivers but not the standard TimeHarp 200 data acquisition software. Installing both the standard TimeHarp software and DLL-based custom programs on the same computer is possible, but only one program at a time can use the TimeHarp board.

To install the DLL and demos, please run the setup program SETUP.EXE in the THLib/installation directory of the CD. If you received the setup files as a ZIP archive, please unpack them to a temporary directory on your hard disk and run SETUP.EXE from there. On some Windows platforms you need administrator rights to perform the setup. If the setup is performed by an administrator but used from other accounts without full access permission to all disk locations, these restricted accounts may not be able to run the demos in the default locations they have been installed to. In such cases it is recommended that you copy the demo directory (or selected files from it) to a dedicated development directory, in which you have the necessary rights (e.g. in ‘My Documents’).

You also need to install the TimeHarp 200 board if you have not done so before (see your TimeHarp manual). The TimeHarp DLL will access the TimeHarp 200 board through dedicated device drivers. You need to make sure the device driver has been installed correctly. Normally this is done when Windows first detects the board. You will then be asked to insert the appropriate driver disk. Both the standard TimeHarp software installation disk and the DLL installation disk contain the drivers. You can use the Windows Device Manager to check if the board has been detected and the driver is installed. On the supported Windows platforms you typically need administrator rights to perform hardware setup tasks. Refer to your TimeHarp 200 manual for other driver installation details.

It is recommended to start your work with the TimeHarp 200 board by using the standard interactive TimeHarp 200 data acquisition software. This should give you a better understanding of the board’s operation before attempting your own programming efforts. It also ensures that your optical/electrical setup is working. If you are planning to use a router, try to get everything working without router first, to avoid additional complications.

5. The Demo Applications - Functional Overview

Please note that all demo code provided is correct to our best knowledge, however, we must disclaim all warranties as to fitness for a particular purpose of this code. It is provided ‘as is’ for no more than explanatory purposes.

The demos are kept as simple as possible to maintain focus on the key issues of accessing the board. This is why most of the demos have a minimalistic user interface and/or run from a simple DOS box (console). For the same reason, the measurement parameters are mostly hard-coded and thereby fixed at compile time. It may therefore be necessary to change the source code and re-compile the demos in order to run them in a way that is matched to your individual measurement setup. Running them unmodified may result in useless data (or none at all) because of inappropriate input level settings etc.

There are demos for C/C++, C#, Delphi, LabVIEW, MATLAB and Visual Basic. For each of these programming languages/systems there are different demo versions for various measurement modes:

Standard Mode Demos

These demos show how to use the standard measurement mode for on-board histogramming. They are provided for all programming languages mentioned above. These are the simplest demos and the best starting point for your own experiments. In case of LabVIEW the standard mode demo is already very sophisticated and allows interactive input of most parameters.

Continuous Mode Demos

These demos show how to use continuous mode, i.e. recording "strings" of histograms with short measurement times and no time gaps between histograms. Because this requires real-time processing and/or real-time storing of data these demos are more demanding both in programming skills and computer performance. See the section about continuous mode in your TimeHarp 200 manual. The continuous mode demos also exist for all programming languages listed above.

TTTR Mode Demos

These demos show how to use TTTR mode, i.e. recording individual photon events instead of forming histograms on board. This mode attaches a 100 ns time tag to each photon record, in addition to the picosecond start-stop time. This permits extremely sophisticated data analysis methods, such as single molecule burst detection or the combination of fluorescence lifetime measurement with FCS. Because TTTR mode requires real-time processing and/or real-time storing of data, these demos are also very demanding both in programming skills and computer performance. See the section about TTTR mode in your TimeHarp 200 manual. TTTR mode always implicitly performs routing if a PRT/NRT 400 router and multiple detectors are used. It is also possible to record external TTL signal transitions in the TTTR data stream (see the TimeHarp 200 manual).

Routing Demos

Multi channel measurement (routing) is possible in standard histogramming mode if a PRT/NRT 400 router and multiple detectors are connected. The routing demos show how to perform such measurements and how to access the histogram data of the individual detector channels. The concept of routing mode is very simple and similar to standard histogramming. If a NRT 400 router is used, the router's internal CFDs must be programmed (TH_SetNRT400CFD). Note that routing is not possible in continuous mode. See the section about routing in your TimeHarp 200 manual.

Imaging with the SCX 200 Scan Controller

The TimeHarp 200 programming library also supports Fluorescence Lifetime Imaging with the SCX 200 scan controller. The SCX 200 is a scanning/imaging controller available as an accessory for the TimeHarp 200 board. It allows users to connect a voltage controlled high resolution 2-D piezo scanner and collect fluorescence lifetime data synchronously with high speed scanning. Scanned data acquisition is performed in TTTR mode. This can be combined with routing. Currently there are no demos for scan operation. Section 7 'Advanced Techniques' provides a guideline for programming the scan controller.

6. The Demo Applications by Programming Language

As outlined above, there are demos for C/C++, C#, Delphi, MATLAB, LabVIEW and Visual Basic. For each of these programming languages/systems there are different demo versions for the measurement modes listed in the previous section. This manual explains the special aspects of using the TimeHarp 200 DLL, it does NOT teach you how to program in the chosen programming language. We strongly recommend that you do not choose a development with the TimeHarp DLL as your first attempt at programming. You can find some general hints on how to call the library routines from the various programming languages in the individual language specific folders (See 'Calling.txt' files). The ultimate reference for details about how to use the DLL is in any case the source code of the demos and the header files of THlib (thlib.h and thdefin.h). Be warned that wrong parameters and/or variables, invalid pointers and buffer sizes, inappropriate calling sequences etc. may crash your application and/or your complete machine. This may even be the case for relatively safe operating systems such as Windows 2000, XP or Vista because you are accessing a kernel mode driver through THLib.dll. This driver has high privileges at kernel level, that provide all power to do damage if used inappropriately. Make sure to backup your data and/or perform your development work on a dedicated machine that does not contain valuable data. Note that the DLL is not re-entrant. This means, it cannot be accessed from multiple, concurrent processes or threads at the same time. All calls must be made sequentially in the order shown in the demos.

The C/C++ Demos

The demos are provided in the 'C' subfolder. The code is actually plain C to provide the smallest common denominator for C and C++. Consult thlib.h, thdefin.h and this manual for reference on the library calls. The library functions must be declared as `extern "C"` when used from C++. This is achieved most elegantly by wrapping the entire include statements for the library headers:

```
extern "C"
{
#include "thdefin.h"
#include "thlib.h"
}
```

In order to make the exports of THlib.dll known to the rest of your application you may use thlib.exp or link directly with the import library thlib.lib. Thlib.lib was created for MSVC 4.0 or higher, with symbols decorated in Microsoft style. MSVC++ users who have version 5.0 or higher can use the supplied project files (*.dsw) where linking with thlib.lib is already set up. The DLL also (additionally) exports all symbols undecorated, so that other compilers should be able to use them conveniently, provided they understand the Microsoft LIB format or they can create their own import library. The MinGW compiler understands the Microsoft format. With Borland C++ 5.x and C++Builder 3.0 you can use the Borland Utility IMPLIB to create your own import library very easily. A ready-to-use 'thlib_bc.lib' can be found in the TH200Lib folder. Failing to work with an import library you may still load the DLL dynamically and call the functions explicitly.

To test any of the demos, consult the TimeHarp manual for setting up your 200 board and establish a measurement setup that runs correctly and generates useable test data. Compare the settings (notably CFD and sync levels) with those used in the demo and use the values that work in your setup when building and testing the demos.

The C demos are designed to run in a console ("DOS box"). They need no command line input parameters. They create their output files in their current working directory (*.out). The output files will be ASCII in case of the standard and routing demos. For continuous and TTTR mode the output is stored in binary format for performance reasons. The ASCII files will contain single or multiple columns of integer numbers representing the counts from the 4096 histogram channels. You can use any editor or a data visualization program to inspect the ASCII histograms. The binary files must be read by dedicated programs according to the format they were written in.

The C# Demos

The C# demos are provided in the 'Csharp' subfolder. They have been tested with MS Visual Studio 2005 as well as with Mono under Windows and Linux. The only difference is the library name, which in principle could also be unified.

Calling a native DLL (unmanaged code) from C# requires the `DllImport` attribute and correct type specification of the parameters. Not all types are easily portable. Especially C strings require special handling. See Calling.txt.

With the C# demos you also need to check wether the hardcoded settings are suitable for your actual instrument setup. The demos are designed to run in a console ("DOS box"). They need no command line input parameters. They create their output files in their current working directory (*.out). The output files will be ASCII in case of the standard and routing demos. For continuous and TTTR mode the output is stored in binary format for performance reasons. The ASCII files will contain single or multiple columns of integer numbers representing the counts from the 4096 histogram channels. You can use any editor or a data visualization program to inspect the ASCII histograms. The binary files must be read by dedicated programs according to the format they were written in.

The Delphi/Lazarus Demos

Delphi and Lazarus users refer to the ‘DELPHI’ directory. Everything for the respective demo is in the project file for that demo (*.DPR). Lazarus users cann access the project through the corresponding *.LPI files. In order to make the exports of THlib.dll known to your application you have to declare each function in your Pascal code as ‘external’. This is already prepared in the demo source code. Thlib.dll was created with MSVC 6.0, with symbols decorated in Microsoft style. It additionally exports all symbols undecorated, so that you can call them from Delphi with the plain function name. Please check the function parameters of your code against THLIB.H in the demo directory whenever you update to a new DLL version. For general reference on calling DLLs from Delphi see CALLING.TXT.

The Delphi/Lazarus demos are also designed to run in a console ("DOS box"). They need no command line input parameters. They create output files in their current working directory. The output files will be ASCII in case of the standard and routing demos. For continuous and TTTR mode the output is stored in binary format for performance reasons. You can use any data visualization program to inspect the ASCII histograms. The binary files must be read by dedicated programs according to the format they were written in.

The Visual Basic Demos

The Visual Basic demos are in the ‘VB’ directory. VB users should start their work in standard mode from DLLDEMO.BAS. The code should be fairly self explanatory. If you update to a new DLL version please check the function parameters of your existing code against THLib.h in the demo directory. Note that where pointers to C-arrays are passed as function arguments you must pass the first element of the corresponding VB array by reference. In the VB function declaration you must declare the argument as the element type, not as an array.

The VB demo programs use a simple console for user I/O, which can be replaced by the usual visual components to build a GUI. For VB there are no pre-compiled EXEs of the demos since you need to have VB installed on your machine anyway, to get the VB runtime libraries. All measurement parameters are set directly in the VB code. However, you should run the standard TimeHarp 200 software first, to see if your hardware setup is correct.

The VB demos (except continuous mode) create output files in their current working directory (*.out). The output file will be ASCII in case of the standard and routing demos. For TTTR mode the output is stored in binary format for performance reasons. The ASCII files will contain columns of integer numbers representing the counts from the 4096 histogram channels. You can use any data visualization program to inspect the histograms. The binary files must be read by dedicated programs according to the format they were written in.

Although the demos were initially written for Microsoft's VB 6.0, they can easily be migrated to Visual Studio 2005/2008. Just use the migration assistant and follow the instructions.

The MATLAB Demos

The MATLAB demos are provided in the ‘MATLAB’ directory. They are contained in m–files. You need to have a MATLAB version that supports the 'calllib' function. We have tested with MATLAB 7.3 but any version from 6.5 should work. Be very careful about the header file name specified in 'loadlibrary'.

This name is case sensitive and a wrong spelling will lead to an apparently successful load but later no library calls will work. The same applies to the chosen alias name.

Calling a DLL from MATLAB is slightly cumbersome when there are parameters passed by reference. See 'Calling.txt' in the 'MATLAB' directory.
The MATLAB demos are designed to run inside the MATLAB console. They need no command line input parameters. They create output files in their current working directory. The output file will be ASCII in case of the histogramming and routing demos. In TTTR and continuous mode the output is stored in binary format for performance reasons. You can use any data visualization program to inspect the ASCII histograms. The binary files must be read by dedicated programs according to the format they were written in. The file read demos provided for the TimeHarp data files can be used as a starting point. Note, however, that they cannot be used directly on the demo output because they expect a file header the demos do not generate. This is intentional in order to keep the demos focused on the key issues of using the library.

The LabVIEW Demos

The LabVIEW demo VIs are provided as LabVIEW in the 'LABVIEW' directory. They are contained in LabVIEW libraries (*.llb). The top level VI is always 'TimeHarp.vi'. Note that the sub-VIs in the various demos are not always identical, even though their names may be the same. You need to have LabVIEW 7.1 or higher. You may initially get warnings with newer LabVIEW versions. These warnings will disappear as soon as you save the project under the new version.

The LabVIEW demos are the most sophisticated demos here. The standard mode demo closely resembles the standard TimeHarp software with input fields for all settable parameters. Run the toplevel VI TimeHarp.vi. It will first initialize and calibrate the hardware. The status of initialization and calibration will be shown in the top left display area. Make sure you have a running TCSPC setup with sync and detector correctly connected. You can then adjust the sync level until you see the expected sync rate in the meter below. Then you can click the *Run* button below the histogram display area. The demo implements a simple *Oscilloscope mode* of the TimeHarp. Make sure to set an acquisition time of not much more than e.g. a second, otherwise you will see nothing for a long time. If the input discriminator settings are correct you should see a histogram. You can stop the measurement with the same (*Run)* button.

The TTTR and continuous mode demos for LabVIEW are a little simpler. They provide the same panel elements for setting parameters etc. but there is no graphic display of results. Instead, all data is stored directly to disk.

To run the continuous mode demo you start TimeHarp.vi. First set up the Sync and CFD levels. You can watch the sync rate in a graphic rate meter. Then you can select a number of banks and a measurement time per block and a file name. When you click the *Run* button a measurement will be performed, with the data going directly to disk. There are status indicators to show the current bank, the counts per bank and the number of lost blocks. There is also a status LED showing any error.

A special note must be made regarding the histogram data in continuous mode: LabVIEW stores binary data always in big endian format. On the x86 platform this results in reversed bytes compared to C programs. This byte reversing of the data going to disk can be avoided by declaring the buffer for TH_GetBank as a byte array (but then 2 times longer). You may also use a byte reverse function (time consuming). If your goal is not storing data to disk but doing some on-line analysis of the histograms it is better to leave the buffer setup as in the demo.

To run the TTTR mode demo you start TimeHarp.vi. First set up the Sync and CFD levels. You can watch the sync rate in a graphic rate meter. Then you can select a measurement time and a file name. When you click the *Run* button a measurement will be performed, with the data going directly to disk. There is a status indicator showing the current number of counts recorded. There is also a status LED indicating any FiFo overrun. Internally the TTTR mode demo also deserves a special note: each TTTR record as returned in the buffer of TH_T3RStartDMA actually is a DWORD (32bit). However, LabVIEW stores DWORD data (U32) always in big endian format. On the x86 platform this results in reversed bytes compared to C programs. For consistency with the demo programs for reading TTTR data this byte reversing of the data going to disk is avoided in the demo by declaring the buffer for TH_T3RStartDMA as a byte array (hence 4 times longer than the DWORD array). You may instead want to work with a U32 array if your goal is not storing data to disk but doing some on-line analysis of

the TTTR records. In this case you must initialize the array with 65536 x U32 and change the type of buffer in the library calls of TH_T3RStartDMA to U32.

The LabVIEW routing demo is very similar to the standard mode demo. It displays 4 traces (for the 4 routing channels) at the same time. There are also four independent numeric displays for the integral count of each channel. This demo requires the PRT/NRT 400 router. It will not run if the router is not connected.

The LabVIEW demos access the DLL routines via the 'Call Library Function' of LabVIEW. For details refer to the LabVIEW application note 088 'How to Call Win32 Dynamic Link Libraries (DLLs) from LabVIEW' from National Instruments. Consult thlib.h or the manual section further down for the parameter types etc. Make sure to specify the correct calling convention (stdcall).

Strictly observe that the TH_xxxx library calls must be made sequentially in exactly the right order. They cannot be called in parallel as is the default in LabVIEW if you place them side by side in a diagram. Sequential execution must be enforced by sequence structures or data dependency. In the demos this is e.g. done by chained and/or nested case structures. This applies to all VI hierarchy levels, so sub-VIs containing library calls must also be executed in correct sequence.

7. Advanced Techniques

Efficient DMA Data Transfer

Some of the TimeHarp measurement modes are designed for fast real-time data acquisition. In particular TTTR mode is most efficient in collecting data with a maximum of information and even from multiple detectors (routing). It is therefore most likely to be used in sophisticated on-line data processing scenarios, where it may be worth optimizing data throughput.

In order to achieve the highest throughput, the TimeHarp 200 uses interrupt controlled busmastering Direct Memory Access (DMA). This means that the board can become the PCI bus master and independently transfer data to the host memory without help of the CPU. This permits data throughput as high as 2 Mcps and leaves time for the host to perform other useful things, such as on-line data analysis or storing data to disk.

In TTTR mode the DMA process is exposed to the DLL user in two functions:

int _stdcall TH_T3RStartDMA(unsigned long* buffer, unsigned long count);
int _stdcall TH_T3RCompleteDMA();

To keep the demos simple and clear, these functions are used in all demos in the following sequence:

1) TH_T3RStartDMA
2) TH_T3RCompleteDMA
3) Store (or process) the data

TH_T3RStartDMA basically tells the TimeHarp board where to put the data and the bus master begins the DMA transfer. TH_T3RCompleteDMA does nothing but wait for the DMA to complete which is signalled by an interrupt from the TimeHarp board. Internally the wait is implemented as the standard Windows function 'WaitForSingleObject' that gives its share of CPU time to other processes or threads i.e. it waits without burning CPU time. This wait time is what should be used for doing 'useful things' in terms of the desired data processing or storing. There are two ways of achieving this. The first is to 'interleave' the DMA process and the data processing. In fact this is the reason why the DMA process was split into two separate routines. The sequence then looks like this:

1) TH_T3RStartDMA
2) Store (or process) the data
3) TH_T3RCompleteDMA

You probably noticed that this creates a problem: When the DMA transfer is not yet completed, how can we already process the data? Right. We cannot. This interlaced sequence only makes sense in a loop where the sequence repeats for several times. However, this is exactly what happens in any TTTR run of significant length. Then the processing step can be applied to the chunk of data from the

previous loop. Obviously this requires at least two data buffers, so that one can be filled and one can be processed. You will also need markers that keep track of which buffer is used at a given time and whether it contains valid data. In no case a buffer must be accessed while a DMA on that buffer is in progress. This method gives throughput improvements between 50 and 100%. However, we have deliberately NOT implemented it in the demos. It is an advanced technique and beginners may run into trouble. Users with sufficient programming skills will be able to implement it from the explanations given here.

The other way of using the wait time during DMA is to use multithreading. In this case you design your program with two threads, one for collecting the data (i.e. working on the TimeHarp board with the DMA functions) and another for processing the data. Of course you need to provide an appropriate data queue between the two threads and the means of thread synchronization. Thread priorities are another issue to be considered. Finally, if your program has a graphic user interface you may need a third thread to respond to user actions reasonably fast. Again, this an advanced technique and it cannot be demonstrated in detail here. Greatest care must be taken not to access the TimeHarp board from different threads without strict control of mutual exclusion and maintaining the right sequence of function calls. However, the technique also allows throughput improvements of 50..100% and advanced programmers may want to use it. It might be interesting to note that this is how TTTR mode is implemented in the standard TimeHarp software, where sustained count rates over 2.5 Mcounts/sec (to disk) can be achieved on modern PCs.

Working with Very Low Count Rates

If high throughput optimizations as outlined above are not of interest, the pair TH_T3RStartDMA and TH_T3RCompleteDMA (immediately one after another) can be replaced by the single call of TH_T3RDoDMA. This is usually only of interest for clarity of the code. Performance is the same. Only in one case TH_T3RDoDMA is especially useful: When working with VERY low count rates it may be undesirable to wait for the FiFo to become half full. Because counts come in very slowly, it can take quite long to fill half the FiFo. In this case, one may want to fetch a single record as soon as the flag FiFoEmpty becomes zero. Fetching single records is, however, much less efficient than doing DMA over large blocks. In order to remove at least the overhead of setting up and waiting for completion of the DMA, TH_T3RDoDMA treats the case with a transfer size (count) of 1 differently: In that case it does not perform DMA, but a single 32-bit input operation via the CPU. This allows to fetch single records at a reasonable throughput rate. Typically count rates up to 100 KHz can be handled this way. Obviously this approach puts a heavier load on the CPU and must be avoided when concurrent data processing is necessary. It should be noted that one can even combine DMA and single record retrieval, dependent on the count rate. For this purpose one would fetch single records as long as FiFoHalfFull remains zero and switch to DMA mode with blocks of half a FiFo size as soon as FiFoHalfFull is one. This leads to a dynamic adaption to the count rate. Again, it must be noted that this an advanced technique that cannot be demonstrated in detail here.

Programming the SCX 200 Scan Controller

The TimeHarp 200 programming library also supports Fluorescence Lifetime Imaging with the SCX 200 scan controller. The SCX 200 is a scanning/imaging controller available as an accessory for the TimeHarp 200 board. It allows users to connect a voltage controlled high resolution 2-D piezo scanner and collect fluorescence lifetime data synchronously with high speed scanning. Simple (slow) movements to any x-y position can be done in histogramming mode and TTTR mode. Scanned (image) data acquisition is performed in TTTR mode only. This can be combined with routing. Programs for automated raster scans should be derived from the TTTR mode demos. Please request direct support if necessary.

The following is a brief outline of the necessary steps:

first, after TH_Init:

TH_ScanEnable(1);

The return code can be used to detect the presence of the SCX 200.

Going to any x-y position and/or open/close the shutter:

TH_ScanGotoXY(x, y, shutter);

For initial testing/debugging it is recommended to connect voltmeters to the control voltage and shutter outputs. These calls can be made repeatedly but for a full image scan they are usually too slow. All movements are made slowly to protect the piezo stage (typically 1 ms per step). Combined x-y movements are always first going along the x axis, then along the y-axis.

Pre-programming fast automatic raster scans (TTTR mode only):

TH_ScanSetup(startx, starty, widthx, widthy, pixtime, pause, shutterdel);

The actual scan begins only after a TTTR mode measurement is started. See the corresponding demo. For initial testing/debugging it is recommended to connect an oscilloscope to the control voltage outputs.

At the end:

TH_ScanDone();

This moves back to the position before the start of scan and restores the previous shutter state.

For a soft return to the zero position after a scan termination at an unknown position:

TH_ScanReset();

8. Data Types

The TimeHarp DLL is written in C and its data types correspond to standard C/C++ data types on 32 bit platforms as follows:

char	8 bit, byte (or character in ASCII)
short int	16 bit, signed integer
unsigned short int	16 bit, unsigned integer
int	32 bit, signed integer
unsigned int	32 bit, unsigned integer
long int	32 bit, signed integer
unsigned long int	32 bit, unsigned integer
float	32 bit, floating point

These types are supported by all major programming languages. Unfortunately Visual Basic is somewhat limited, it does not have the 16-bit types. You need to use byte arrays and convert each pair of successive bytes to the equivalent 16 bit number in a 32 bit variable.

9. Functions exported by THLIB.DLL

See **thdefin.h** for predefined constants given in capital letters here. Return values <0 usually denote errors. See **errcodes.h** for the error codes. ALL functions must be called with *_stdcall* convention.

```
int TH_GetErrorString(char* errstring, int errcode);

arguments:     errstring = pointer to a buffer for at least 40 characters
               errcode = error code returned from a TH_xxx function call

return value:  >0      success
               <0      error

note: This function is provided to obtain readable error strings that explain
the cause of the error better than the plain error code. Use these in all
error handling message boxes etc.
```

```
int TH_GetLibraryVersion(char* vers);

arguments:     vers = pointer to a buffer for at least 7 characters

return value:  =0      success
               <0      error

note:          this is the only function you may call before TH_Initialize
```

```
int TH_Initialize(int mode);

arguments:     mode    0=standard histogramming, 1=TTTR

return value:  =0      success
               <0      error
```

```
int TH_GetHardwareVersion(char* vers);

arguments:     vers = pointer to a buffer for at least 7 characters

return value:  =0      success
               <0      error
```

```
int TH_GetSerialNumber(char* serial);

arguments:     vers = pointer to a buffer for at least 7 characters

return value:  =0      success
               <0      error
```

```
int TH_GetBaseResolution(void);

arguments:     void

return value:  >0      approximate base resolution of the installed board
               <0      error

note: This function is provided to obtain an approximate value of the baord BEFORE
calibration. For exact values, please calibrate first and then use TH_GetResolution.
```

```
int TH_Calibrate(void);

arguments:     void
```

```
return value:  =0     success
               <0     error
```

int TH_SetCFDDiscrMin(int value);

```
arguments:     value = CFD discriminator level in millivolts
                                  minimum = DISCRMIN
                                  maximum = DISCRMAX
return value:  =0     success
               <0     error
```

int TH_SetCFDZeroCross(int value);

```
arguments:     value =  CFD zero cross level in millivolts
                                  minimum = ZCMIN
                                  maximum = ZCMAX

return value:  =0     success
               <0     error
```

int TH_SetSyncLevel(int value);

```
arguments:     value = Sync level in millivolts
                                  minimum = SYNCMIN
                                  maximum = SYNCMAX

return value:  =0     success
               <0     error
```

int TH_SetStopOverflow(int StopOnOfl, int StopLevel);

```
arguments:     StopOnOfl  (0 = do not stop, or 1 = do stop on Overflow)
               StopLevel   count level at which to stop (min 1, max 65535)

return value:  =0     success
               <0     error

note:          This setting determines if a measurement run will stop if any channel
               reaches the stop level.
```

int TH_SetRange(int range);

```
arguments:     value = Measurement range code
                           minimum = 0           (smallest, i.e. base resolution)
                           maximum = RANGES-1    (largest)

return value:  =0     success
               <0     error

note:          range code 0 = base resolution, 1 = 2 x base resolution and so on.
```

int TH_SetOffset(int offset);

```
arguments:     value =        Offset in nanoseconds
                                  minimum = OFFSETMIN
                                  maximum = OFFSETMAX

return value:  >=0 new offset
               <0 error

note:          The new offset is an approximation of the desired offset
               The typical step size is around 2.5 ns
```

```
int TH_NextOffset(int direction);

arguments:     value = direction of the desired offset change
                               minimum = -1 (down)
                               maximum = +1 (up)

return value:  >=0 new offset
               <0 error

note:          The offset changes at a step size of approximately 2.5 ns


int TH_ClearHistMem(int block);

arguments:     block = block number to clear (always 0 if not routing)

return value:  =0      success
               <0      error


int TH_SetMMode(int mmode, int tacq);

arguments:     mmode = 0 for one-time histogramming and TTTR
                       1 for continuous mode

               tacq =  acquisition time in milliseconds
                       minimum = ACQTMIN
                       maximum = ACQTMAX
                       set this to 0 for continuous mode with external clock

return value:  =0      success
               <0      error


int TH_StartMeas(void);

arguments:     void

return value:  =0      success
               <0      error


int TH_StopMeas(void);

arguments:     void

return value:  =0      success
               <0      error

note:          In TTTR mode this resets the hardware FiFo


int TH_CTCStatus(void);

arguments:     void

return value:  0       acquisition time still running
               >0      acquisition time has ended


int TH_SetSyncMode(void);

arguments:     void

return value:  =0      success
               <0      error

note:          This function must be called before TH_GetCountRate()
               if the sync rate is to be measured. Allow at least 500ms
               after this call to get a stable sync rate reading.
```

int TH_GetBlock(unsigned int *chcount, int block);

```
arguments:     *chcount =     pointer to an array of double words (32bit) of BLOCKSIZE
                              where the histogram data can be stored
               block = block number (0..3) to fetch (always 0 if not routing)

return value:  >=0     total number of counts in this histogram
               <0      error

note:          The current version counts only up to 65535 (16 bits).
               This may change in the future.
```

float TH_GetResolution(void);

```
arguments:     void

return value:  >0      resolution (channel width) in current range (from last calibration)
               <0      error
```

int TH_GetCountRate(void);

```
arguments:     void

return value:  >=0     current count rate or sync rate (see note)
               <0      error

note:          The function returns either the current count rate if previously
               TH_SetMmode() was called or the current sync rate if previously
               TH_SetSyncMode() was called. Allow at least 500ms to get a stable
               rate meter reading in both cases.
```

int TH_GetFlags(void);

```
arguments:     void

return value:  current status flags (a bit pattern)

note:          Use the predefined macros in thdefin.h (e.g. FLAG_OVERFLOW)
               to extract individual bits through a bitwise AND.
               You can call this function anytime during measurement but not
               during DMA.
```

int TH_GetElapsedMeasTime(void);

```
arguments:     void

return value:  elapsed measurement time in ms

note:          Not for continuous mode.
```

int TH_Shutdown();

```
arguments:     void

return value:  >0      success
               <0      error

note:          Closes the device.
```

SPECIAL FUNCTIONS FOR CONTINUOUS MODE

int TH_GetBank(unsigned short *buffer, int chanfrom, int chanto);

```
arguments:     *buffer        pointer to an array of words (16bit) of BANKSIZE
                              where the histogram data can be stored

               chanfrom
               chanto         limits of the channels to be fetched per histogram
                              (min 0, max 4095, and chanfrom<=chanto)

return value:  >0      sum of all counts in the channels fetched
               <0      error

note:          Chanfrom and chanto are included in the channels that are fetched.
               The number of valid channels in buffer will therefore be (chanto-chanfrom)+1
               The channel specified by chanfrom will be placed in buffer location 0.
```

int TH_GetLostBlocks(void);

```
arguments:     void

return value:  >0      number of lost blocks in continuous mode run
               <0      error

note:          The lost blocks will be counted to a maximum of 255, then remain at 255.
```

SPECIAL FUNCTIONS FOR TTTR MODE (Must have TTTR mode option)

int TH_T3RDoDMA(unsigned int* buffer, unsigned int count);

```
arguments:     *buffer        pointer to an array of dwords (32bit) of at least ½ FIFOSIZE
                              where the TTTR data can be stored
               count          number of TTTR records to be fetched (max ½ FIFOSIZE)

return value:  =0      success
               <0      error

note:          Initiates and performs DMA in a single function call.
               Must not be called with count larger than buffer size permits.
               CPU time during wait for DMA completion will be yielded to other
               processes/threads. Function will return after a timeout period of 300 ms, even
               if not all data could be fetched. Buffer passed to TH_T3RDoDMA must not be
               accessed until the function returns.
```

int TH_T3RStartDMA(unsigned int* buffer, unsigned int count);

```
arguments:     *buffer        pointer to an array of dwords (32bit) of at least ½ FIFOSIZE
                              where the TTTR data can be stored
               count          number of TTTR records to be fetched (max ½ FIFOSIZE)

return value:  =0      success
               <0      error

note:          Initiates DMA but returns before completion.
               Must not be called with count larger than buffer size permits.
               No other calls to THlib are allowed between this call and the return
               of a subsequent call to TH_T3RCompleteDMA.
```

int TH_T3RCompleteDMA(void);

```
arguments:     void
```

```
return value:  >=0     number of records transfered
               <0      error

note:          Waits for DMA to complete. CPU time during wait will be yielded to other
               processes/threads. Function will return after a timeout period of 300 ms, even
               if not all data could be fetched. Buffer passed in TH_T3RStartDMA must not be
               accessed until TH_T3RCompleteDMA returns.
```

int TH_T3RSetMarkerEdges(int me0, int me1, int me2);

```
arguments:     me0, me1, me2          0 = falling edge, 1 = rising edge

return value:  =0      success
               <0      error

note:          Sets the active edges of the TTL signals that will appear as markers in
               the TTTR data stream.
```

SPECIAL FUNCTIONS FOR ROUTING (Must have PRT/NRT 400 Router)

int TH_GetRoutingChannels(void);

```
arguments:     void

return value:  >0      number of routing channels available (1 = no router)
               <0      error
```

int TH_EnableRouting(int enable);

```
arguments:     enable         0=disable, 1=enable

return value:  =0      success
               <0      error (most likely: no router found)
```

int TH_SetNRT400CFD(int channel, int level, int zerocross); (Must have NRT 400 Router)

```
arguments:     channel        0..3
               level                  discriminator level, 0..400 (mV)
               zerocross              zero cross voltage 0..40 (mV)

return value:  =0      success
               <0      error (most likely: no NRT 400 router present)
```

SPECIAL FUNCTIONS FOR SCANNING (Must have SCX 200 Controller)

int TH_ScanEnable(int enable);

```
arguments:     enable         0=disable, 1=enable

return value:  =0      success
               <0      error (most likely: no scan controller found)
```

int TH_ScanGotoXY(int x, int y, int shutter);

```
arguments:     x              x co-ordinate to move to (0..4095)
               y              y co-ordinate to move to (0..4095)
               shutter        shutter output TTL high/low (1/0)

return value:  =0      success
               <0      error
```

```
int TH_ScanDone(void);

arguments:     void

return value:  =0      success
               <0      error


int TH_ScanSetup(int startx, int starty, int widthx, int widthy,
                            int pixtime, int pause, int shutterdel);

arguments:     startx         x co-ordinate start of scan region (0..4095)
               starty         y co-ordinate start of scan region (0..4095)
               witdhx         x co-ordinate width of scan region (2..4096)
               witdhy         y co-ordinate width of scan region (2..4096)
               pixtime        pixel dwell time in µs (100..3200)
               pause          pause at turning points of x axis (0..15)
               shutterdel     shutter delay to start of scan in ms (0..1000)

return value:  =0      success
               <0      error

note:          During scan: x = fast axis, y = slow axis.
               Actual scan is performed in next run of TTTR mode.


int TH_ScanReset(void);

arguments:     void

return value:  =0      success
               <0      error

note:          This call is only required at program start or to recover from an
               unexpected error or crash.
```

10. Problems, Tips & Tricks

PC Performance Issues

Performance issues with the DLL are the same as with the standard TimeHarp software. The TimeHarp 200 board and its software interface are a complex real-time measurement system demanding considerable performance both from the host PC and the operating system. This is why a reasonably modern CPU (500 MHz min.) and at least 512 MB of memory are required. However, as long as you do not use continuous or TTTR mode, these issues should not be of severe impact. If you do intend to use continuous or TTTR mode you should also have a fast modern hard disk and possibly use a faster PC with 2 GHz CPU clock or better and 2048 MB of memory. The latter is recommended anyhow, if you use Windows Vista.

Troubleshooting

Troubleshooting should begin by testing your hardware and driver setup. This is best accomplished by the standard TimeHarp software for Windows (supplied by PicoQuant). Only if this software is working properly you should start work with the DLL. If there are problems even with the standard software, please consult the TimeHarp manual for detailed troubleshooting advice.

The TimeHarp DLL will access the TimeHarp 200 board through dedicated device drivers. You need to make sure the device driver has been installed correctly. Normally this is done when Windows first detects the board. You will then be asked to insert the appropriate driver disk. Both the standard TimeHarp software installation disk and the DLL installation disk contain the drivers. You can use the Windows Device Manager to check if the board has been detected and the driver is installed. Note that you may need to perform a driver update (via Device Manager) when you upgrade from an earlier Version of the TimeHarp software or DLL. On the supported Windows platforms you need administrator rights to perform hardware setup tasks. Please consult the TimeHarp 200 manual for hardware related problem solutions.

The next step, if hardware and driver are working, is to make sure you have the right DLL version installed. It comes with its own setup program that must be executed as Administrator. In the Windows Explorer you can also right click THLib.DLL (in Windows/System or Windows/System32) and check the version number (under Properties). You should also make sure your board has the right firmware to use the DLL. If not, you can order an upgrade. Then try the readily compiled demos supplied with the DLL (installed under 'THLib Utilities And Demos'). For first tests take the standard C demo (dlldemo.exe). If this is working, your own programs should work as well.

Note that the DLL will not allow to start two or more instances of any software using the DLL and/or the standard TimeHarp software on one machine. Since there is only one piece of hardware, this would lead to conflicts during hardware access, which cannot be allowed. However, the DLL may not be aware of running instances of older versions of the TimeHarp software or older DLL versions. Therefore, you must ensure that you do not run older TimeHarp software at the same time you run your custom program. It is recommended to upgrade the driver, the DLL and the standard software to the most recent version.

Warming Up and Calibration

Please observe the warming up and calibration recommendations given for the standard TimeHarp software also in your own developments. Inform the user to allow a warming-up period of at least 10 minutes before using the TimeHarp for serious measurements. It is possible to use this time for set-up and preliminary measurements. Calibrate one more time before starting serious measurements. For very long measurements allow some more time for thermal stabilisation and avoid temperature changes >10°C in the environment of the PC. The operating temperature in the PC must not exceed 50°C. Make sure that the cooling fan of the PC is operating correctly and the cooling air can circulate freely.

File Access Permissions

On all recent Windows platforms you need administrator rights to perform the THLib setup. If the setup is performed by an administrator but used from other accounts without full access permission to all disk locations, these restricted accounts may no be able to run the demos in the default locations they have been istalled to. In such cases it is recommended that you copy the demo directory or selected

files from it to a dedicated development directory in which you have the necessary rights. Otherwise the administrator must give full access to the demo directory. On Windows XP and Vista it is possible to switch between user accounts without shutting down the running applications. It is not possible to start a TimeHarp program if any other program accessing the board is running in another user account that has been switched away. Doing so may cause crashes or loss of data.

Version Tracking

While PicoQuant will always try to maintain a maximum of continuity in further hardware and software development, changes for the benefit of technical progress cannot always be avoided. It may therefore happen, that data structures, calling conventions or program flow will change. In order to design programs that will recognize such changes with a minimum of trouble we strongly recommend that you make use of the functions provided for version retrieval of hardware and DLL. In any case your software should issue a warning if it detects versions other than those it was tested with. In favour of compatibility with existing software the driver DLL for the TimeHarp 200 board bears the same name as that for the TimeHarp 100 board. The version numbers in these two systems are independent. In the unlikely case that you use both board versions you need to keep the two DLLs in separate places. In case of doubt about any given DLL file, use the built in file properties (right click the file and select Properties) to identify the version. All future TimeHarp 100 versions will bear version numbers <4.0 and all future TimeHarp 200 versions will bear version numbers >=4.0

Support

The TimeHarp 200 TCSPC system has gone through several iterations of hardware improvement and extensive testing. Nevertheless, it is a fairly challenging development and some glitches may still occur under the myriads of possible PC configurations and application circumstances. We therefore offer you our support in any case of trouble with the system and ask for your help to identify any such problems. Do not hesitate to contact PicoQuant in case of difficulties with your TimeHarp board or the DLL. Should you observe errors or unexpected behaviour related to the TimeHarp system, please try to find a precise and reproducible error situation. E-mail a detailed description of the problem and relevant circumstances to *info@picoquant.com.* Please include the version information of driver and DLL. Also provide all PC system details, ideally obtained through *msinfo32*. Your feedback will help us to improve the product and documentation.

If you have serious problems that require the board to be sent in for inspection/repair, please request an RMA number before sending the hardware. Unexpected shippings will be returned. Observe precautions against damage, especially static discharge, under all circumstances in handling, packaging and shipping. Inappropriate packaging may void warranty and insurance. Also make sure customs forms are filled in correctly, so that a return shipping for repair is clearly identified as such. Neglecting this may lead to double import taxation upon return.

More likely you will actually have good news. Let us know! Many of our customers have used the instruments with great success in leading edge research. If you have obtained exciting results with one of our systems, please let us know, and where appropriate, mention us in your publications. At our Web-site we maintain a bibliography of hundreds of publications related to our instruments and research. See http://www.picoquant.com/_biblio.htm. Please submit your references for addition to this list.

PicoQuant GmbH
Unternehmen für optoelektronische Forschung und Entwicklung
Rudower Chaussee 29 (IGZ), 12489 Berlin, Germany

Tel: +49 / (0)30 / 6392 6560
Fax: +49 / (0)30 / 6392 6561
e-mail: info@picoquant.com
WWW: http://www.picoquant.com