

Mail Addresses:

Headquarter

RoentDek Handels GmbH
Im Vogelshaag 8
D-65779 Kelkheim-Ruppertshain
Germany

Frankfurt subsidiary

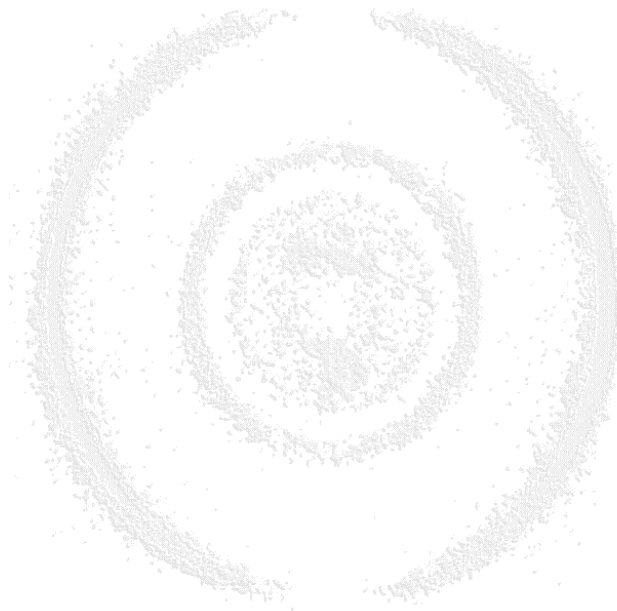
RoentDek Handels GmbH
c/o Institut für Kernphysik
Max-von-Laue Str. 1
D-60438 Frankfurt am Main
Germany

Web-Site:

www.roentdek.com

WEEE:

DE48573152



Product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

All rights reserved. Technical changes may be made without prior notice. The figures are not binding.

We make no representations or warranties with respect to the accuracy or completeness of the contents of this publication

Table of Contents

1	THE HARDWARE.....	5
1.1	FEATURES.....	5
1.2	INTENDED USE.....	5
1.3	CONCEPTS	6
1.4	INPUTS.....	6
1.5	ROLLOVER.....	6
1.6	TRIGGER.....	6
1.7	GROUPS	7
1.8	SIGNAL LEVEL INFORMATION.....	7
1.9	DATA FORMAT	8
1.9.1	<i>Falling transition</i>	8
1.9.2	<i>Rising transition</i>	8
1.9.3	<i>Error with channel information</i>	8
1.9.4	<i>Group</i>	9
1.9.5	<i>Rollover</i>	9
1.9.6	<i>Level information</i>	9
1.9.7	<i>Resolution</i>	9
1.10	CONFIGURATION FILE.....	9
1.10.1	<i>Values</i>	9
1.10.2	<i>Boolean</i>	10
1.10.3	<i>Integer</i>	10
1.10.4	<i>Time</i>	10
1.10.5	<i>Channel mask</i>	10
1.10.6	<i>Edge</i>	10
1.10.7	<i>Parameters</i>	10
1.10.8	<i>Parameters configurable by the user</i>	10
1.10.9	<i>Deprecated or Obsolete Parameters</i>	11
1.11	MULTIPLE BOARDS	11
1.11.1	<i>Multiple boards with external clock reference</i>	12
1.12	CALIBRATION	12
1.13	PROGRAMMING INTERFACE	12
1.13.1	<i>Exceptions</i>	12
1.13.2	<i>Startup and cleanup</i>	13
1.13.3	<i>Configuration</i>	13
1.13.4	<i>Reflection</i>	13
1.13.5	<i>Control</i>	14
1.13.6	<i>Readout</i>	16
1.14	TDC DRIVER STATE MACHINE.....	16
1.15	CODE EXAMPLE.....	18
1.15.1	<i>C++ Code Example</i>	18
1.15.2	<i>C# Code Example</i>	18
1.16	ELECTRICAL INPUT CHARACTERISTIC	20
1.16.1	<i>Hi-Res Inputs</i>	20
1.16.2	<i>Trigger Input</i>	20
1.16.3	<i>Clock Input</i>	20
1.16.4	<i>Lowres Input</i>	21
1.16.5	<i>TDC8 Sync Input</i>	21
1.17	TIME MEASUREMENT PARAMETERS	22
1.18	DATA RATE	22
1.19	RECYCLING.....	23
2	TDC8HP PCI CARD INSTALLATION.....	24
2.1	TDC8HP PCI DOUBLE CARD INSTALLATION	24
2.2	DRIVER INSTALLATION.....	25
2.2.1	<i>Driver Installation via SetupCD Program</i>	26
2.2.2	<i>Manual Driver installation.</i>	26

2.2.3 Driver uninstallation..... 26

2.2.4 Manual Driver uninstallation. 26

3 USING THE TDC8HP SYSTEM 28

3.1 C SAMPLE SOURCE CODE FOR TDC8HP INITIALIZATION AND READ-OUT 28

3.2 USING THE TDC8HP CARD WITH COBOLDPC DAQ SOFTWARE..... 29

3.2.1 DAn and DAq Modules 29

3.2.2 The "Standard.ccf" 29

3.2.3 DAq parameters 30

3.2.4 Additional DAq parameters for TDC8HP..... 30

3.2.5 DAq coordinates 34

3.2.6 DAn parameters and coordinates: 34

3.2.7 Spectra and conditions..... 34

4 KNOW BUGS 36

LIST OF FIGURES 38

LIST OF TABLES 38



1 The hardware

The **TDC8HP** system is based on the CERN HPTDC chip. The **TDC8HP** system consists of the TDC8HP board and the **CoboldPC** software. This card supports nearly all features of the MTD133B chip.

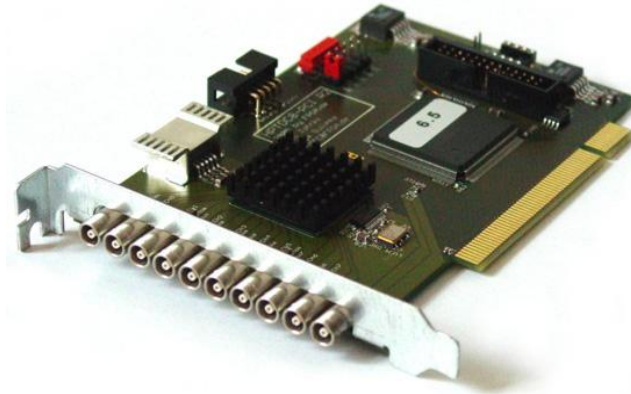


Figure 1.1: TDC8HP PCI Card

1.1 Features

- 8 NIM compatible inputs on LEMO connectors with 25ps bin size
- 1 NIM compatible input on LEMO connector with 12.8ns bin size
- 12 LVCMOS inputs on SUB-D connector with 12.8ns bin size
- typical deadtime between multiple hits on one channel <5ns
- unlimited number of hits per trigger
- no dead time due to readout, new data is acquired during readout
- maximum of 2M Hits/s readout rate
- 419 μ s (extendable) range w. trigger logic enabled, 62ms range in extended mode
- 2h range without trigger logic, can be extended by software (not yet implemented)
- adjustable trigger window (size, position of trigger)
- easy to use driver for windows operating systems (C++ and DotNet Framework)
- on board storage for calibration data
- support for up to three event-synchronized boards
- 5V, 32-bit, 33MHz PCI target device

1.2 Intended Use

Typical applications for a TDC8HP-PCI include atomic physics experiments (e.g. momentum imaging, time-of-flight spectroscopy), mass spectroscopy and LIDAR.

The TDC8HP-PCI may not be used

- for medical applications except for
 - research
 - imaging systems
 - medical devices used solely as diagnostic tools
- in military devices
- in conjunction with nuclear materials related to defense or power systems
- for space applications with the exception of fundamental research

The TDC8HP-PCI may not be used for military purposes.

If the TDC8HP-PCI is sold, the above restrictions must be incorporated in any resale contract.

1.3 Concepts

The TDC8HP-PCI continuously records the digital waveforms on its inputs similar to a logic analyzer with an accuracy of up to 25ps (least significant bit (LSB)).

1.4 Inputs

The board has nine AC coupled inputs on LEMO series 00 coax connectors compatible to the NIM signaling standard. Eight of these are high resolution inputs and one is a low resolution input. Furthermore, it has twelve DC coupled low resolution inputs for 3.3V LVCMOS signals (LVTTL compatible).

The NIM inputs are terminated with 50 Ohms to ground. A current of -16mA (equivalent to an input voltage of -800mV) is identified as a logic 1, and no current (equivalent to an input voltage of 0V) is identified as a logic 0.

The LVCMOS inputs identify voltages above 2V as logic 1 and voltages below 0.8V as logic 0.

The high resolution inputs have a bin size of 25ps; the low resolution inputs have a bin size of 12.8ns. In order to simplify analysis, high and low resolution inputs use the same data formats. The time for all channels is stored as integer multiples of a unit time, usually 25ps.

Transitions of the input signals are called hits. The data acquisition can be limited to rising or falling signal transitions. Channels can be enabled individually. The high resolution channels are numbered 0 to 7, the optional DC coupled low resolution channels are numbered 9 to 20 and the single AC coupled low resolution channel has channel number 8.

For use with
CoboldPC

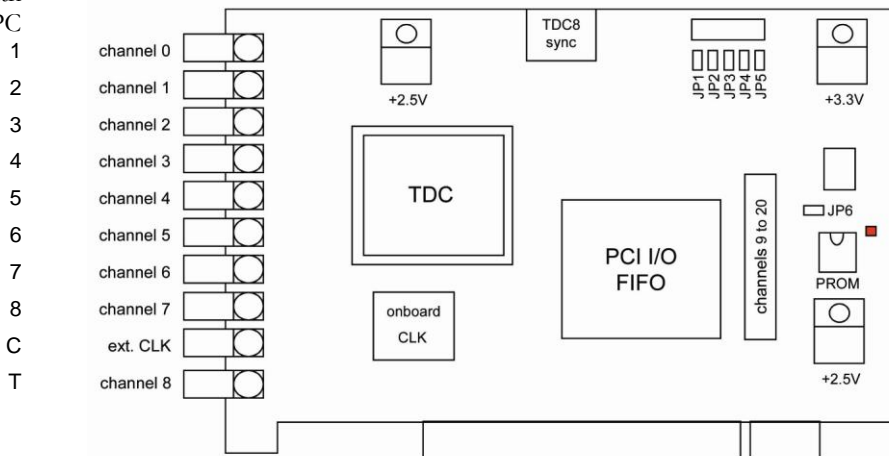


Figure 1.2: The TDC8HP-PCI card. The high resolution channels 0-7 and the low resolution NIM-channel 8 are located on the left. Low resolution channels 9-20 are connected to the 26-pin ribbon cable connector on the right side. The 10-pin ribbon cable connector at the top is used to connect and synchronize a TDC8PCI2-card to the TDC8HP-PCI.

1.5 Rollover

The data format for a hit contains 24 bits of timing information, which is sufficient for a 419µs time interval. The TDC8HP-PCI is collecting data continuously. Whenever the 24 bit counter rolls over a rollover marker is inserted into the data stream with an additional 24 bits for the time measurement (thus allowing measurements of up to ~2h). The time interval between two rollovers is called a frame. If there is no hit between two consecutive rollover markers only the last marker appears in the output. However, it is guaranteed that at least one rollover is output before the 48 bit counter overflows. Using this information measurements with infinite range can be performed if grouping is disabled.

The rollover markers can optionally be suppressed by the driver if grouping is enabled in order to simplify the data format and reduce the data rate for applications that do not need timing information that spans multiple groups. Without grouping enabled rollovers are always output.

1.6 Trigger

An arbitrary input channel can be selected as trigger input. A trigger condition occurs for a signal transition on that input. The time of the trigger condition can be used to group the input data (see below).

Triggers can be rearmed automatically after a programmable delay, after the group associated with the trigger is finished or manually.

1.7 Groups

Hits can be merged to groups (or events in HEP nomenclature). A group will contain hits that occurred within a certain time interval relative to a trigger. Both, the start and end point of the interval can be configured within a range of $-209.7\mu\text{s}$ to $209.7\mu\text{s}$. Figure 1.3 shows a typical data acquisition scenario in triggered mode. The size of the acquisition window is set by the configuration parameters *GroupRangeStart* and *GroupRangeEnd*. The falling edge of the trigger signal is used (*TriggerEdge falling*).

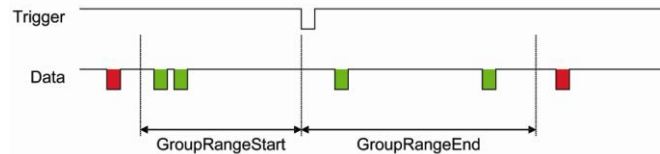


Figure 1.3: Data acquisition in triggered mode. Hits within the group's range (green) are acquired; hits outside that range (red) are rejected.

Data within a group has timing values relative to the trigger event's time. Hits can be assigned to multiple groups. If a new trigger event occurs inside the range of a group one of three behaviors can be selected:

1. No trigger is issued and the trigger event is recorded as a hit on the trigger channel if data taking for this channel is enabled.
2. A new trigger is issued, the previous group is ended and data in the overlapping range is assigned to the new group.
3. A new trigger is issued and data in the overlapping range are assigned to both groups possibly including the trigger event.

The range of a group is limited to $419.4\mu\text{s}$.

Figure 1.4 displays the three types of behavior. On top, a trigger is suppressed using the configuration parameter *TriggerDeadTime*. In the middle of Figure 1.4, a group is ended by a consecutive trigger. A new group is started. Both groups' ranges are truncated. This behavior is enabled using the configuration parameter *AllowOverlap false*. Figure 1.4 (bottom) shows the case of overlapping groups (*AllowOverlap true*). Hits inside the overlapping region of both groups are assigned to each group, the timing value is calculated with respect to the corresponding trigger time.

If grouping is disabled the timing value for a hit is an unsigned integer relative to the last rollover marker.

A group in the data stream ends with the beginning of a new group or with a rollover marker.

If grouping and rollovers are enabled there is a rollover marker immediately preceding each group marker, even if two consecutive groups start in the same frame.

As described before, a trigger deadtime can be set in order to suppress consecutive triggers in the chosen timing interval.

This can be used to avoid overlapping groups or create an intentional dead time to remove unwanted data from the data stream.

1.8 Signal Level Information

(only for the 12 LVCMOS inputs on SUB-D connector)

If grouping of data is enabled, input transitions that occur outside a group are not recorded. If the signal has no transition within the group the level of the signal is unknown. Therefore the signal levels can be output at the beginning of each group. This feature can be enabled individually for each channel.

If multiple TDCs are used the level information is not guaranteed to appear directly at the beginning of a group for all TDCs. Level information for each board is guaranteed to appear before the first signal transition of that board.

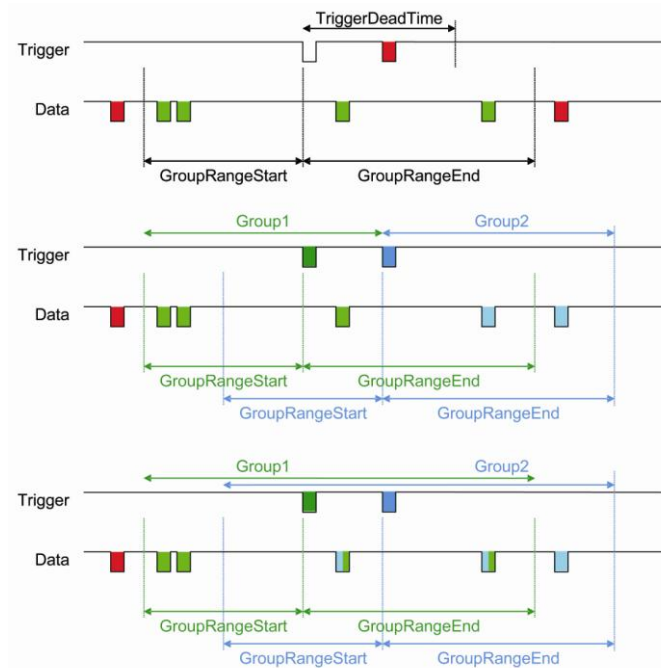


Figure 1.4: The three different types of grouping described in the text

1.9 Data Format

The readout data is a stream of 32-bit words in little endian byte order. Each word represents a signal transition on an input or a special marker.

1.9.1 Falling transition

31-30	29 - 24	23 - 0
10	Channel	transition time

Table 1.1: Falling transition

The transition time is a signed integer when inside a group or an unsigned integer otherwise

1.9.2 Rising transition

31-30	29 - 24	23 - 0
11	channel	transition time

Table 1.2: Rising transition

The transition time is a signed integer when inside a group or an unsigned integer otherwise

1.9.3 Error with channel information

31-30	29 - 24	23 - 16	15 - 0
01	channel	error	count

Table 1.3: Error with channel information

Error numbers below 128 are lost hits. Error numbers where the bit with value 64 is set should reset the data acquisition if multiple boards are present and the trigger channel is in the group the error occurred in

Error Number	Source	Description
0 (0x00)	FPGA	highres data lost due to fpga fifo overflow
16 (0x10)	SOFT	data lost due to software buffer overflow
32 (0x20)	FPGA	lowres data lost due to fpga fifo overflow
96 (0x60)	FPGA	triggers lost due to fpga fifo overflow

112 (0x70)	SOFT	trigger lost due to software buffer overflow
160 (0xa0)	TDC	TDC error, reset acquisition
128 (0x80)	FPGA	unknown error, should occur only for prototype
129 (0x81)	FPGA	FPGA fifo empty
255 (0xff)	SOFT	reset because boards might be out of sync

Table 1.4: Error numbers

1.9.4 Group

31 - 28	27 - 24	23 - 0
0000	Id	trigger time

Table 1.5: Group

The time values of the following hits should be added to the trigger time to obtain the absolute time of the transition. The id currently is always 0 but could be used to identify multiple trigger conditions in future versions.

1.9.5 Rollover

31 - 24	23 - 0
00010000	upper 24 bits of time stamp

Table 1.6: Rollover

The time between two rollovers is called a frame

1.9.6 Level information

31 - 27	26 - 21	20 - 0
00011	n	Level

Table 1.7: Level Information

At the beginning of a group the levels of the input signals can optionally be output. One word contains the levels of 21 inputs beginning at channel n

1.9.7 Resolution

31 - 24	23 - 0
00010000	Bin size in fs (25000)

Table 1.8: Resolution

This word should be added once at the beginning of each output file to describe the size of a timing bin. This way in the future multiple TDC types can be supported or an INL-lookup that changes the bin size can be performed. The resolution marker might be repeated in arbitrary intervals. If the resolution marker is missing a bin size of 25ps should be assumed

1.10 Configuration File

TDC, FPGA and Driver are configured by text based configuration files that can also contain calibration data.

Any number of configuration files can be read in with the newer configuration overriding the older ones. This allows to have configuration files that each describe individual boards, experiment setups and readout situations and combine them as desired.

Configuration files consist of independent lines that can appear in any order. Each line configures zero or one parameter. If the same parameter appears multiple times in a set of configuration files the last occurrence is used.

Lines starting with a sharp „#“ are comments and are ignored by the driver.

Anything behind a double slash “//” to the end of a line is treated as a comment.

There may be any number of empty lines.

All other lines consist of a parameter name and a value. Both, values and parameter names, are case insensitive.

1.10.1 Values

Values can have different types as follows. If no value is specified a default value is loaded from the onboard PROM.

1.10.2 Boolean

A value that is either true or false. The values “1”, “P”, “true”, “on”, “enable” and “enabled” are recognized as true. The values “0”, “F”, “false”, “off”, “disable” and “disabled” are recognized as false.

1.10.3 Integer

Any integer representation that is legal in C such as „123“, “0xff” or “-1”.

1.10.4 Time

A floating point number in C-notation such as “5”, “5.2”, “1.7e-3” followed by a time unit. Allowed are “s”, “ms”, “us”, “µs”, “ns”, “ps” and “fs”.

1.10.5 Channel mask

A value of this type defines a set of channels that the parameter applies to. The format is a list of ranges or individual channels separated by commas. A range consists of a lower and upper channel separated by a minus. The lower and upper bound are included in the range. Examples are „1“ or „7, 9, 14“ or „1-4,15“. To include all channels use „0-20“. To disable all channels use „no“ or „none“

1.10.6 Edge

An input transition. The value “falling” denotes a transition from a higher electrical potential to a lower potential. “rising” denotes the opposite transition.

1.10.7 Parameters

Parameters are named with a case insensitive string of alphabetic characters.

Array parameters are followed by “:” and an index.

If multiple TDCs are present most parameter names can be followed by “@” and a number indicating for which TDC the parameter shall be set. Otherwise the parameter is set for all TDCs. Numbering of TDC boards starts with 0.

Additionally, for parameters that can be set channelwise, the name can be followed by “#” and a number to indicate for which channel the parameter shall apply. Otherwise the parameter is applied for all channels.

1.10.8 Parameters configurable by the user

name	type	scope	min	max	default	description
RisingEnable	mask	channel			none	Record 0 to 1 transitions for these channels
FallingEnable	mask	channel			0-63	Record 1 to 0 transitions for these channels
TriggerEdge	edge	global			falling	Trigger on falling or rising edge
TriggerChannel	int	board	0	63	0	Which channel to use as a trigger
OutputLevel	bool	board	false	true	false	Keep track of signal levels
GroupingEnable	bool	global	false	true	true	Enable grouping
AllowOverlap	bool	global	false	true	false	If a hit falls within multiple groups copy it to all groups or just to the first one
TriggerDeadTime	time	global	0	1s	0	Time during which no new triggers are accepted after a trigger
GroupRangeStart	time	global	-209.7µs	209.7µs	209µs	Smallest time value relative to group marker to be included in group. (Set to minus range size for common stop)
GroupRangeEnd	time	global	-209.7µs	209.7µs	209.7µs	Largest time value relative group marker to be included in group. (Set to plus range size for common start)
ExternalClock	bool	global	false	true	false	Use external clock source
OutputRollovers	bool	global	false	true	true	Include rollover markers in the data format. Rollovers can only be disabled if grouping is enabled.
VHR	bool	global	false	true	true	Enables the TDC's very high resolution mode with a bin size of 25ps. If disable the bin size is set to 100ps increasing the TDC's data throughput by a factor of 4, thus increasing

						lossfree hit rates (see section 1.18).
UseFineINL	bool	global	false	true	false	Use better INL correction with ReadTDCHit(). Disables coarse INL correction. Should not be set when using Read().
GroupTimeout	time	global	0	100	0.2	Timeout in seconds for the Read-function. If smaller than 1ms then value is treated as 0s
BufferSize	int	global	16	27	23	Number of bits in the number of hits in the buffer. 23 Bits provides a buffer of 8MHits
DITapAdjust:0 to DITapAdjust:31	int	board	0	7	From PROM	Lookup table for DNL correction?
DelayTap:0 to DelayTap:3	int	board	0	7	From PROM (7,7,0,0)	Calibrate very high resolution clock
INL:0 to INL:1023	int	channel	0	1023	From PROM	Lookup table for INL correction
UseClock80	bool	global	false	true	true	
MMXEnable	bool	global	false	true	true	Use 64 bit reads in non-dma mode
DMAEnable	bool	global	false	true	true	Copy data using direct memory access
SSEEnable	bool	global	false	true	false	USE 128 bit reads in non-dma mode

Table 1.9: Parameter configurable by the user

1.10.9 Deprecated or Obsolete Parameters

These parameters have been valid in old versions of the driver but should not be used anymore.

SoftwareSync	bool	global	false	true	false	Enable if more than one card is used, but no external clock source is present.
TDC8Sync	bool	global	false	true	false	Synchronize a TDC8 and a TDC8HP-PCI board.
SyncValidationChannel	int	board	0	20	0	
SimulateExternalClock	bool	global	false	true	false	

Table 1.10: Deprecated or Obsolete Parameters

1.11 Multiple boards

When using multiple boards simultaneously it is required that they are plugged into the same physical PCI bus.

Unique channel numbers are assigned to each board. The second board will usually start with channel number 21 and the third one with channel number 42. A serial number of the type X.Y is printed on every TDC card (see Figure 1.5). X denotes the year of production minus 2000, Y is an integer greater than zero. The card with the lowest number is initialized as the first board, the one with the next higher number as the second and the board with the highest number is the third board. For example: three cards with the numbers 6.7, 7.11 and 6.12 are installed. The first card (with channel numbers 0 to 20) is the one with 6.7, the second card (channels 21 to 41) is the one with serial number 6.12 and the third card (channels 42 to 62) is card 7.11.



Figure 1.5: The card's serial number located on the PCI-interface chip. The first digit denotes the year of production minus 2000, the second is a unique serial number.

Multiple boards with common trigger

When no common clock signal is available a common trigger signal must be connected to all boards. In this case multiple TDC8HP-PCIs can only be used with grouping enabled (if grouping is disabled, only data from the first TDC is acquired). The time stamps of hits inside the group are reported relative to the trigger of the board the hit occurred on. The input stream of each board contains rollovers but only one of these is included in the output stream. There is an additional measurement uncertainty of one bin plus 20ppm when measuring differences between the two boards. The configuration parameter *SoftwareSync* enables or disables this mode of data acquisition.

1.11.1 Multiple boards with external clock reference

This is the preferred mode of operation when using multiple boards: a common clock signal is connected to the clock input of each board.

In this case all acquisition modes described above are supported and the additional measurement error described above is avoided.

Using an external clock, a trigger signal needs to be connected to one TDC board only. This must be the *first* board (channels 0 to 20).

In order to enable the external clock source the configuration parameter *ExternalClock* needs to be set to *true*.

1.12 Calibration

There are two levels of calibration: first, for the very high resolution mode of the TDC8HP-PCI the bin size for the lower two bits of measurement can be adjusted using delay taps inside the TDC. Second, the INL of the data can be corrected using a table lookup in the driver software. It is an option to use a different bin size after table lookup than before table lookup. For the table lookup the last ten bits of each timing value are used to index a table that contains 1024 entries for each channel with the last bits of the corrected measurement.

The boards are delivered with calibration data stored in an onboard flash PROM. The driver uses this data by default. Users can easily create their own calibration data and feed it to the driver or even program new data to the onboard prom.

When using the `Read()` method data is returned with 25ps quantization with both corrections applied.

When using the `ReadTDCHits()` method data is returned with 1ps quantization. The average bin size will still be 25ps, but the information about the individual bin sizes can be preserved. To use this feature set the “useFineINL” property to “true”.

1.13 Programming Interface

The driver API defines the following methods. Both the C++ and the C# syntax is shown. The C# version can be used in any language that supports the .net platform such as VisualBasic, J# or managed C++.

C++: displays the C++ function definition

C#: displays the c# .net platform function definition

1.13.1 Exceptions

Many methods in the API throw exceptions when an error occurs. They all have the type `TDCConfigException` which is defined as:

```
class TDCConfigException {
public:
    const char * errorString;
};
```

The error string contains additional information about the nature of the exception.

The application developer can choose to catch the exception and continue with program execution. Optionally the error string could be logged or presented to the user.

If the exception is not caught the application will be terminated.

1.13.2 Startup and cleanup

1.13.2.1 Init

C++: void Init()

C#: void Init()

Should be called once before using the TDCManager. *Init()* detects all present TDC devices, acquires all available information on the those devices and maps the address space for PCI access. It also reads the „global.cfg“ configuration file if present in the current directory.

Can throw all exceptions that are related to parsing the configuration file.

1.13.2.2 CleanUp

C++: void CleanUp()

C#: void CleanUp()

Should be called once before exiting the program.

1.13.2.3 GetTDCCount

C++: int GetTDCCount()

C#: int GetTDCCount()

Returns the number of TDC8HP boards found in the system. From 0 to 3.

1.13.3 Configuration

The TDC and readout configuration is manipulated via a set of methods. The parameters do not take effect before the *Reconfigure()* method is called.

If a parameter is set multiple times the last value is used.

1.13.3.1 SetParameter & ReadConfigString

C++: bool SetParameter(const char *config)

C#: bool SetParameter(string config managed)

C++: bool ReadConfigString(const char * parameter)

C#: bool ReadConfigString(string configManaged)

Read in a string with the syntax of a configuration file line as described in the previous section 1.10. The string might contain multiple lines.

Can only be called if the TDC is stopped (states 0, 1 and 2). Throws an exception otherwise.

Returns false if there were illegal parameters or syntax errors. True otherwise.

C++: bool SetParameter(const char *property, const char * value)

C#: bool SetParameter(string propManaged, string valueManaged)

Set the value of a parameter identified by the parameter name.

Can only be called if the TDC is stopped (states 0, 1 and 2). Throws an exception otherwise.

Returns true if successful, false if there was an error.

1.13.3.2 Reconfigure

C++: void Reconfigure()

C#: void Reconfigure()

Writes configuration data to the device. Use after configuration has been changed. This operation is slow.

Can only be called if the TDC is stopped (states 0, 1 and 2). Throws an exception otherwise.

Throws an exception if a parameters cannot be parsed..

1.13.4 Reflection

1.13.4.1 GetParameter

C++: const char * GetParameter(const char * parameter)

C#: String GetParameter(string parameterManaged)

Get the value of a parameter. If the parameter does not exist or has not been set an empty string is returned.

1.13.4.2 GetParameterNames

C++: `const char ** GetParameterNames(int & count)`

C#: `string[] GetParameterNames()`

Get the names of all parameters that are set. The c++ version returns the number of elements in the array in the count variable.

1.13.4.3 GetDriverVersion

C++: `int GetDriverVersion()`

C#: `int GetDriverVersion()`

The lowest three bytes returned are the three digits of the driver Version.

1.13.5 Control

1.13.5.1 Start

C++: `void Start()`

C#: `void Start()`

Configure all TDCs with the parameters set by the methods described in the previous section and start all TDCs. This method can require several milliseconds to complete when called from state "NOT_CONFIGURED". To precisely time the start of the data acquisition call *reconfigure()* first.

If called from state NOT_CONFIGURED exceptions thrown from *reconfigure()* can occur.

Also throws an exception if called from state UNINITIALIZED.

1.13.5.2 Stop

C++: `void Stop()`

C#: `void Stop()`

Stop taking data and turn off TDCs. The state of the hardware and software queues is undefined after this operation. No additional data should be read out. To get all data in the queues call *Pause()* and empty the buffers before calling this method.

The TDC is put into a low power state that can only be recovered by a *Start()* operation.

Throws an exception if called in states UNINITIALIZED or SHUTDOWN.

1.13.5.3 Pause

C++: `void Pause()`

C#: `void Pause()`

Stop taking data. Data already in the hardware and software buffers is left intact and can be read out. The TDC hardware is kept in a state that allows to resume data acquisition immediately. Frame counters continue to count.

Throws an exception when called from a state other than PAUSED or RUNNING.

1.13.5.4 Continue

C++: `void Continue()`

C#: `void Continue()`

Quickly continue taking data after a *Pause()*. Buffers are left intact and frame counters are left unaltered.

It is legal to use *Start()* to resume operation instead.

Throws an exception when called from a state other than PAUSED or RUNNING.

1.13.5.5 ClearBuffer

C++: `void ClearBuffer()`

C#: `void ClearBuffer()`

clears all buffers. Only meaningful in state PAUSED.

Throws an exception if called in state RUNNING. Has no effect in other states.

1.13.5.6 GetTDCInfo

C++: `TDCInfo GetTDCInfo(int index)`

C#: `HPTDCInfo GetTDCInfo(int index)`

Get the information on TDC card number index.

Throws an exception if called in state UNINITIALIZED.

Returns an object that contains the following data

1.13.5.7 class TDCInfo or HPTDCInfo

int index;

int channelStart;

```
int channelCount;
int highResChannelCount;
int highResChannelStart;
int lowResChannelCount;
int lowResChannelStart;
double resolution;
DWORD serialNumber;
int version;
int fifoSize;
int *INLCorrection;           //in .net : int[] INLCorrection;
unsigned short *DNLDData;    //in .net: short[] DNLDData;
bool flashValid;
int bufferSize;
```

1.13.5.7.1 index

The index of this board, e.g. 0 for the first board.

1.13.5.7.2 channelStart

The number of the card's first channel as initialized by the driver. For example: if two TDC8HP-PCI card's are present, the second card's channel numbers are usually 21 to 41. That's card first channel is therefore channel number 21.

1.13.5.7.3 channelCount

Total number of channels.

1.13.5.7.4 HighResChannelCount

Number of channels with high resolution.

1.13.5.7.5 highResChannelStart

Number of first channel with high resolution.

1.13.5.7.6 lowResChannelCount

Number of channels with low resolution.

1.13.5.7.7 lowResChannelStart

Number of first channel with low resolution.

1.13.5.7.8 resolution

The TDC card's bin size of the high resolution channels.

1.13.5.7.9 serialNumber

The card's unique serial number. The highest byte denotes the year of production minus 2000, the 3 lower bytes a serial number.

1.13.5.7.10 version

The lowest byte describe the board revision, the next higher byte the firmware revision.

1.13.5.7.11 fifoSize

The size of the onboard PCI-FIFO in data words.

1.13.5.7.12 *INLCorrection

Pointer to an array containing the current INL correction table. The array has a size of 8 x 1024 entries, starting with channel 0, value 0, followed by channel 0, value 1 etc.

1.13.5.7.13 *DNLDData

Pointer to an array containing the bin sizes before INL correction. These are relative sizes that must be normalized to the sum of values for each channel.

1.13.5.7.14 *flashValid*

Set to true if valid calibration flash content has been detected

1.13.5.7.15 *bufferSize*

Size of the software buffer in hits.

1.13.5.8 GetState

C++: `int GetState()`

C#: `int GetState()`

Returns the current state of the TDCManager.

```
const static int STATE_UNINITIALIZED = 0;
const static int STATE_NOT_CONFIGURED = 1;
const static int STATE_CONFIGURED = 2;
const static int STATE_RUNNING = 3;
const static int STATE_PAUSED = 4;
const static int STATE_SHUTDOWN = 5;
```

1.13.6 Readout

1.13.6.1 Read

C++: `int Read(HIT *out, int size)`

C#: `int Read(int[] buffer)`

Copy TDC data into buffer of size count. If grouping is enabled one group is read. Otherwise all available data up to the size of the buffer is read. The number of data words that were read is returned as an integer.

If grouping is enabled and no group is found within a certain time interval *Read()* returns 0.

The data returned has the format described earlier in this document.

1.13.6.2 ReadTDCHit

C++: `int ReadTDCHit(TDCHit *buffer, int length)`

C#: `int ReadTDCHit(HPTDCHit buffer)`

Copy TDC data into buffer of size count. The number of data words that were read is returned as an integer. All available data up to the size of the buffer is read.

Uses a data format that is easier to use and provides a better DNL and INL than *Read()*.

Uses more memory and CPU cycles.

Does not support grouping.

Output data is sorted by timestamp.

The data returned uses a structure that obsoletes rollovers. Also, times are reported in multiples of one picosecond. A more fine-grained INL correction is used in this mode slightly increasing the resolution of the TDC.

```
struct TDCHit {
    public:
        const static int RISING = 1;
        const static int FALLING = 0;
        const static int TDC_ERROR = 2;
        long long time;
        unsigned char channel;
        unsigned char type;
};
```

1.14 TDC Driver State Machine

A sketch of the user view of the driver's architecture is shown in Figure 1.6. After starting the PC the driver will wait in the state Idle until the user invokes an Init command. The Init command loads the parameters given in the standard configuration file („global.cfg“) to the driver's registers. At this point, depicted in Figure 1.6 by the state Not configured, the TDC-card itself is not yet configured. The user can now change any parameter by using the functions *ReadConfigFile* or *SetParameter*. The user may now invoke a Start command in order to start acquiring data. In this case the driver will automatically configure the TDC the card using the parameters set by the user and run the data acquisition. As configuring

the TDC card takes up to 1 second, a second way of starting the data acquisition is implemented, as well: by using the Reconfigure the TDC card is configured and then ready for data acquisition (state Configured in Figure 1.6). A Start command will now start data acquisition instantaneously. This path is intended for users who need a low latency between an intended start and the real start of the data acquisition.

The command *Pause* will stop the data acquisition but leave the TDC card fully configured. Being in the *Pause*-state a *Continue* will immediately restart the data acquisition. If the user intends to change some of the TDC's parameters or to clear the buffers a *Stop* command is necessary taking the driver back to the state *Configured*. If the parameters remain unchanged, a *Start* will immediately restart the data acquisition. By changing parameters the driver returns to the *Not Configured* state, a restart from that state will take longer as the TDC card needs to be reconfigured before the data acquisition is invoked. The current state can be queried with the *getState()* method.

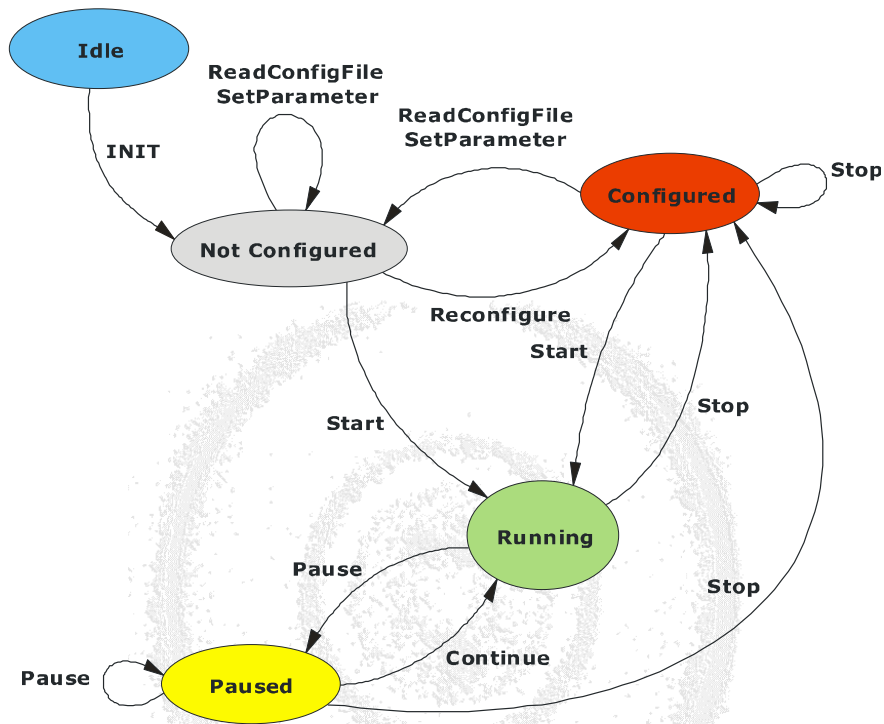


Figure 1.6: The driver's state machine as seen by the user.

1.15 Code Example

1.15.1 C++ Code Example

25ps mode

```
#include "tdcmanager.h"
#include <string>
#include <list>
#include <fstream>
int main(int argc, char* argv[])
{
    TDCManager manager(0x1A13, 0x0001);
    try {
        manager.Init();
        manager.ReadConfigFile("myexperiment.cfg");
        manager.Start();
    } catch (TDCConfigException e) {
        return -1;
    }

    int amountToRead = 1000000;
    HIT buffer[2000];
    ofstream of("test.dat", ios::out | ios::binary);
    of.write(&(0x200061A8), 4); //25ps resolution

    while(amountToRead > 0)
    {
        int count = manager.Read(buffer,2000);
        for(int i = 0; i < count && i < amountToRead; i++) {
            of.write((const char*)(buffer +i), sizeof(HIT));
        }
        amountToRead -= count;
    }
    manager.Stop();
    manager.CleanUp();
    return 0;
}
```

1.15.2 C# Code Example

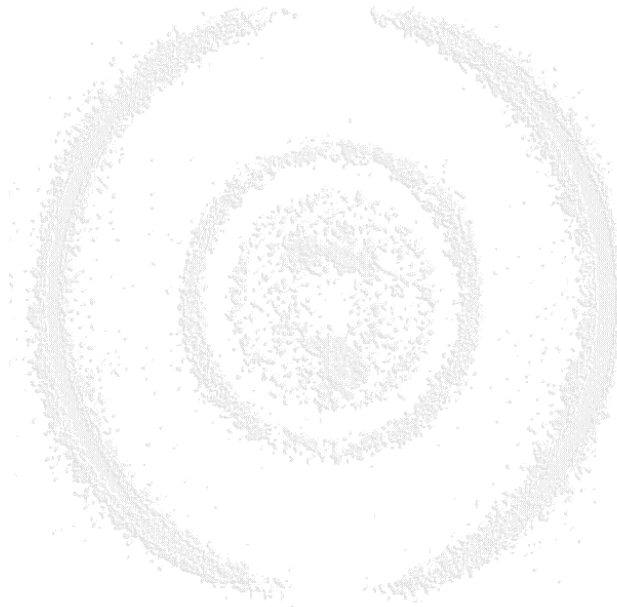
1ps mode

```
using cronologic;
namespace application {
    class Program {
        static void Main(string[] args)
        {
            HPTDCManager manager = new HPTDCManager();
            try {
                manager.Init();
                manager.ReadConfigFile("myexperiment.cfg");
                manager.Start();
            } catch (TDCConfigException e) {
                System.Console.WriteLine(e.errorString);
                return -1;
            }

            int amountToRead = 1000000;
            HPTDCHit[] buffer = new HPTDCHit[2000];

            StreamWriter newFile = new StreamWriter("myexperiment.csv");
```

```
while(amountToRead > 0)
{
    int count = manager.ReadHPTDCHit(buffer);
    for(int i = 0; i < count && i < amountToRead; i++) {
        newFile.Write(buffer[i].channel + ", ");
        newFile.Write(buffer[i].type + ", ");
        newFile.WriteLine(buffer[i].time);
    }
    amountToRead -= count;
}
newFile.Close();
manager.Stop();
manager.CleanUp();
return 0;
}
}
```



1.16 Electrical Input Characteristic

1.16.1 Hi-Res Inputs

AC coupled NIM compatible input's with parallel termination.
The idle level is high, signals are given by negative pulses.

Symbol	Parameter	Min	Typ	Max	Units
	DC input voltage	-10		10	V
	AC Pulse low level (relative to DC level)	-1.5		-0.5	V
	AC switching threshold (relative to DC level)	-0.34	-0.24	-0.14	V
	termination voltage		0		V
	input impedance		50		Ω
	pulse width	5		3000	ns
	dead time after edge		5	10	ns

Table 1.11: High Resolution Inputs

1.16.2 Trigger Input

AC coupled NIM compatible input with parallel termination.
The idle level is high, signals are given by negative pulses.

Symbol	Parameter	Min	Typ	Max	Units
	DC input voltage	-10		10	V
	AC pulse low level (relative to DC level)	-1.5		-0.5	V
	AC Switching threshold (relative to DC level)		-0.38		V
	input impedance		50		Ω
	termination voltage		0		V
	pulse width	30		3000	ns
	pulse separation	30			ns

Table 1.12: Trigger Input

1.16.3 Clock Input

DC coupled input with thievenin termination.

Symbol	Parameter	Min	Typ	Max	Units
	input high level	1.8		2.9	V
	input low level	0.4		1.5	V

	switching threshold		1.65		V
	termination voltage		1.65		V
	input impedance		50		Ω
	clock period	12.8	12.8	13.1	ns
	duty cycle	45	50	55	%

Table 1.13: Clock Input

1.16.4 Lowres Input

DC coupled CMOS compatible inputs with parallel termination.

Symbol	Parameter	Min	Typ	Max	Units
	input high level	2		3.3	V
	input low level	0		0.8	V
	switching threshold		1.65		V
	termination voltage		0		V
	input impedance		100		Ω

Table 1.14: Low Resolution Input

1.16.5 TDC8 Sync Input

DC coupled CMOS compatible inputs with parallel termination.

Symbol	Parameter	Min	Typ	Max	Units
	input high level	2		3.3	V
	Input low level	0		0.8	V
	switching threshold		1.65		V
	termination voltage		0		V
	input impedance		1000		Ω

Table 1.15: TDC8 Sync Input

1.17 Time Measurement Parameters

Symbol	Parameter	Up to 7.x			8.x and up			Units
		Min	Typ	Max	Min	Typ	Max	
	nominal bin size		25			25		ps
	stability over temperature (0°C to 85°C)			20				ppm
	stability over temperature (-20°C to 70°C)						1	ppm
	aging first year				5		1	ppm
	aging thereafter				2		1	ppm/a
	initial frequency inaccuracy						4	ppm
	peak to peak jitter		25					ps
	differential nonlinearity	-25		25	-25		25	ps
	integral nonlinearity	-25		25	-25		25	ps
	Crosstalk (7 aggressors) ¹				150			ps
	Skew between channels ¹				2.4			ns

Table 1.16: Time Base

¹ Not tested

1.18 Data Rate

In order to guarantee that absolutely no hits are lost during data acquisition, the following hit counts must not be exceeded. In many cases exceeding these values will result only in a small amount of lost hits depending on the pattern of the arriving hits.

Symbol	Scope	Number of Hits	Timeframe
	highres channel ¹	1	10ns
	highres channel ¹ (25ps resolution)	4	400ns
	highres channel pair ¹ (100ps resolution)	4	100ns
	highres channel pair ²	256	25µs
	all highres channels ²	512	8µs
	lowres channel ¹	1	30ns
	lowres channel ¹	16	800ns
	all channels ³	2000	500µs

Table 1.17: Data Rate

- ¹ Hits lost as these constraints are not met cannot be detected by hardware
- ² Hits lost due to these constraints are reported as errors in the data stream. The exact number of lost hits cannot be detected.
- ³ Hits lost due to these constraints are reported with the exact number of lost hits. The number of lost hits is reported separately for trigger channels.

1.19 Recycling

Cronologic is registered with the „Stiftung Elektro-Altgeräte Register“ as a manufacturer of electronic systems with Registration ID DE 77895909.

The HPTC belongs to category 9, „Überwachungs und Kontrollinstrumente für ausschließlich gewerblich Nutzung“. The last owner of a TDC8HP must recycle or treat the board in compliance with §11 and §12 of the german ElektroG.



2 TDC8HP PCI card installation

- Shut down your computer
- For your devices safety, turn off the power to your computer and all peripheral devices.
- Drain static electricity from your body by touching the metal chassis (the unpainted metal at the back of your computer)
- For your personal safety, remove the power cord from your computer
- Remove the cover of the computer as described in your computer's manual.
- Locate a free PCI slot in your computer, and firmly insert the card into the selected slot. To avoid damaging your hardware, insert the card only into a slot with the same bus type as the card. Inserting the card into any other type of slot can damage your card, your computer, or both.
- Firmly secure the adapter with a screw (or clip), to ensure that the adapter is properly grounded to the computer's chassis.
- Replace the cover of the computer as described in your computer's manual.

2.1 TDC8HP PCI double card installation

- Shut down your computer
- For your devices safety, turn off the power to your computer and all peripheral devices.
- Drain static electricity from your body by touching the metal chassis (the unpainted metal at the back of your computer)
- For your personal safety, remove the power cord from your computer
- Remove the cover of the computer as described in your computer's manual.
- Locate two free PCI slot in your computer, and firmly insert the cards into the selected slots. To avoid damaging your hardware, insert the cards only into slots with the same bus type as the card. Inserting the cards into any other type of slot can damage your card, your computer, or both.
- Firmly secure the adapters with a screw (or clip), to ensure that the adapters are properly grounded to the computer's chassis.
- Apply the flat ribbon cable to the adapters (connector is on top of the cards).
- Locate a free position for the external clock module. This module is not connected to any bus. Only power is to be supplied to a standard "drive power adapter".

- Connect each “C” channel of the TDC8HP cards to any free port of the external clock using short LEMO cables.



Figure 2.1: External clock module

- Replace the cover of the computer as described in your computer’s manual.

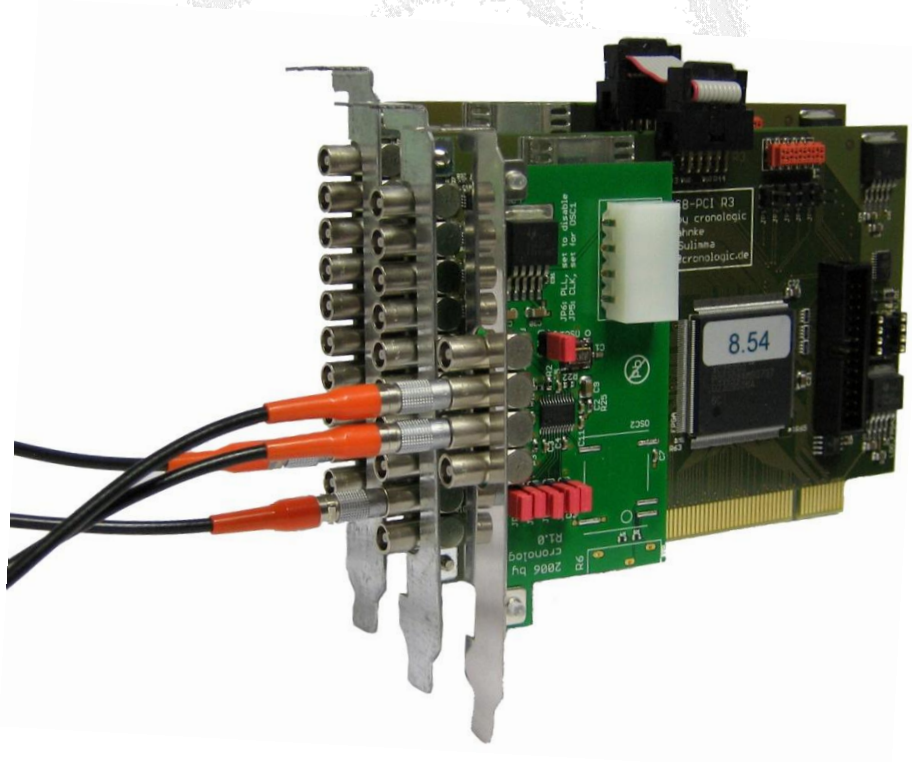


Figure 2.2: Two TDC8HP cards connected with the external clock module.

2.2 Driver Installation

Please install the TDC8HP board(s) prior to driver installation!

After installing the TDC you might get the “Hardware Wizard” when you log into your system (with an “Administration Account”). Please let the System look for Drivers on the CD.

If this procedure fails you may use one of the following methods:

The driver installation procedure described below will install driver version 3.0.2.0. This can be installed parallel to older TDC8HP driver installations.

It is observed, that the driver installation may last long if an anti-virus software is installed. So it is recommended to disable any anti-virus software temporarily for this driver installation.

There will be no entry in the “installed programs” list!

2.2.1 Driver Installation via SetupCD Program

When you insert the **CoboldPC** CD your computer should start automatically the “AutoStart.cmd” program. If not please do so manually.

First it will be checked if DotNet Framework 3.5 SP1 is installed. If it is not installed you’ll get the chance to do it now. This version of the DotNet Framework is needed by the installation program. If it is not installed the start of the CoboldPC2008Setup.Net.exe will result in an error message and will therefore not start.

If the CoboldPC2008Setup.Net.exe is started you’ll find an “Install TDC8HP Driver” button. The “Install TDC8HP Driver” button indicates by colors whether the TDC8HP driver components are installed or not. The red color indicates missing or not installed components. Green indicates that all components are installed correctly.

2.2.2 Manual Driver installation.

- Double click the “DriverInstall.cmd” file and follow the instructions.
- Alternatively open a “Command Prompt” console and switch to the “\Drivers\HPTDC\Install” folder on your CD drive. Then enter the following command lines:

```
wdreg_gui -inf windrvr6.inf install  
wdreg_gui -inf hptdc_windriver.inf install
```

Then copy the files WDAPI920.DLL and HPTDC_DRIVER_3.0.2.DLL to your “system32” directory.

Close your “Command Prompt” console and restart your computer.

2.2.3 Driver uninstallation.

To uninstall the driver automatically, double click the “AutoStart.cmd” from CD to start the setup program. If the button “Install TDC8HP Driver” is green then press it to initiate the uninstall procedure and follow the instructions. If the button is red then the driver is already uninstalled.

2.2.4 Manual Driver uninstallation.

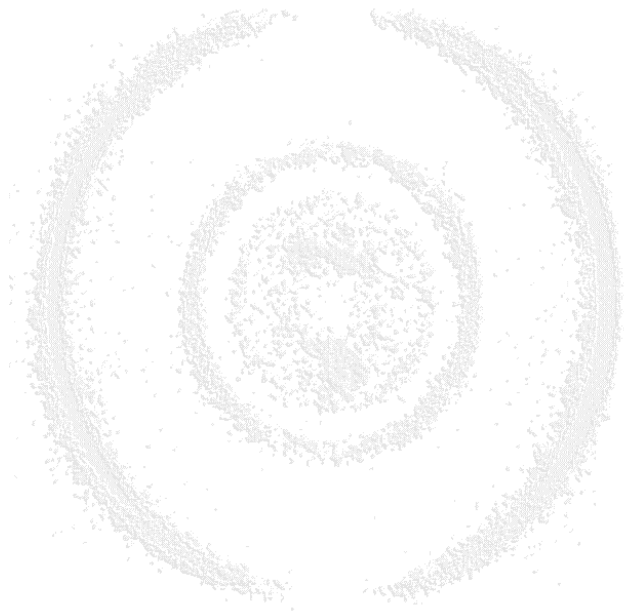
- Double click the DriverUnInstall.cmd file and follow the instructions.
- Alternatively open a “Command Prompt” console and switch to the “\Drivers\HPTDC\Install” folder on your CD drive. Then enter the following command lines:

```
wdreg_gui -inf hptdc_windriver.inf uninstall  
wdreg_gui -inf windrvr6.inf uninstall
```

Then delete the files WDAPI920.DLL and HPTDC_DRIVER_3.0.2.DLL from your “system32” directory.

Close your “Command Prompt” console and restart your computer.

Now start the registry editor (“regedit.exe”) and delete, if present) the following key:
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\WinDriver6



3 Using the TDC8HP System

The TDC8HP card comes with a driver for Windows NT, Windows 2000, Windows XP, Windows 2003 and Windows Vista. The driver consists of the following components.

- Driver to operate the TDC8HP card
- WDAPI920.DLL
- HPTDC_DRIVER_3.0.2.DLL HPTDC_DRIVER_3.0.2.LIB and tdcmanager_3.0.2.h (driver access libraries, link library and header file)

There is program support for the Microsoft Visual C++ compiler (Version .Net 2008 or later). The program interface is described in section 1.13. In Section 1.14 and 3.1 you will find a small sample C program.

3.1 C Sample Source Code for TDC8HP initialization and read-out

The source is not ready for compilation. It is just an incomplete example of how access the TDC8HP card. The code is based on the DAQ module DLL source code for the TDC8HP of CoboldPC.

```
#include <HighResolutionTimer.h>
#include "tdcmanager_3.0.2.h"

////////////////////////////////////
//////// SPECIAL STUFF
////////////////////////////////////
#define MAXIMUM_NUMBER_OF_HITS      16
#define MAXIMUM_NUMBER_OF_CHANNELS  8

TDCManager *manager;
HIT *buffer;
// TDC8HP config parameters
unsigned int uiRisingEnable;           // Parameter 61 (bitmask)
unsigned int uiFallingEnable;         // Parameter 62 (bitmask)
signed int iTriggerEdge;              // Parameter 63 (-1,0,1)
unsigned int uiTriggerChannel;        // Parameter 64 (0-20)
bool bOutputLevel;                   // Parameter 65 (false = 0)
bool bGroupingEnable;                // Parameter 66 (false = 0)
bool bAllowOverlap;                  // Parameter 67 (false = 0)
double dTriggerDeadTime;              // Parameter 68 (time in ns, < 1s)
double dGroupRangeStart;             // Parameter 69 (time in ns, -209µs..+209µs)
double dGroupRangeEnd;               // Parameter 70 (time in ns, -209µs..+209µs)
bool bExternalClock;                 // Parameter 71 (false = 0)
bool bOutputRollovers;               // Parameter 72 (false = 0)
int iDelayTap[4];                    // Parameter 73-76 (0..7)

////////////////////////////////////
//////// SPECIAL STUFF
////////////////////////////////////

bool TDC8HPInit()
{
    // Getting TDC Module
    manager = new TDCManager(0x1a13, 0x0001);
    manager->Init();
    TDCInfo myTDCInfo;
    myTDCInfo = manager->getTDCInfo(0);

    buffer = (HIT *)new HIT[100000];
    bool bConfigFileRead = manager->ReadConfigFile("ConfigFile.cfg");
    if(bConfigFileRead)
    {
        manager->Reconfigure();
        manager->Start();
    }

    return bConfigFileRead;
}

void TDC8HPExit()
{
    manager->Stop();
    manager->Cleanup();
}
```

```
delete (TDCManager*)manager;
delete[] (HIT*)buffer;
return true;
}

int PCIGetTDC_TDC8HP(CDoubleArray *pParameter)
{
    memset(_TDC,0,sizeof(_TDC));           // clear TDC array

    //-----
    // Read one Event, count = number of hits
    //-----
    unsigned int count = 1;
    count = manager->Read(buffer,10000);
    if(!count)
        return false;
    //-----

    bool bOKFlag = false;
    // now process the data
    for(unsigned int i = 0; i < count ; i++)
    {
        if( (buffer[i]&0xC0000000)>=0x80000000)    // valid data only if rising or alling
                                                    // trigger indicated
        {
            lTDCData = (buffer[i]&0x00FFFFFF);
            if(lTDCData & 0x00800000)            // detect 24 bit signed flag
                lTDCData |= 0xff000000;        // if detected extend negative
                                                    // value to 32 bit
            ucTDCChannel = (unsigned char)((buffer[i]&0x3F000000)>>24); // extract channel
                                                    // information

            // calculate TDC channel to _TDC channel
            if((ucTDCChannel >= 42) && (ucTDCChannel <= 49))
                ucTDCChannel -= 25;
            if((ucTDCChannel >= 21) && (ucTDCChannel <= 29))
                ucTDCChannel -= 12;

            if(ucTDCChannel < MAXIMUM_NUMBER_OF_CHANNELS) // if detected channel fits
                                                            // into TDC array then sort
            {
                // increase Hit Counter;
                _TDC[ucTDCChannel][0]++;

                // test for oversized Hits
                if(_TDC[ucTDCChannel][0] > MAXIMUM_NUMBER_OF_HITS)
                    _TDC[ucTDCChannel][0]--;
                Else
                {
                    // if Hit # ok then store it
                    _TDC[ucTDCChannel][_TDC[ucTDCChannel][0]] = lTDCData;
                    bOKFlag = true;
                }
            }
        }
    }

    return true;
}
```

3.2 Using the TDC8HP card with CoboldPC DAQ software

3.2.1 DAn and DAq Modules

To operate your TDC8HP modules please load the appropriate DAn and DAq module DLL files in **CoboldPC** (menu -> File). They are typically placed in the folder installation directory of **CoboldPC**. Make sure that there is also the file HPTDC_DRIVER_3.0.2.DLL in the same folder. It will be needed during the dynamic load of the DAq module.

3.2.2 The "Standard.ccf"

This command script provides already most of the desired data, i.e. 2d position spectra and time-of-flight spectra in various coordinate representations.

The "Startup.ccf" setup assumes that the trigger channel is channel 8. Channel 1 to 7 can be used normally.

restart (reset of earlier commands)

new (defines the session type (online/offline), calls selector box)

start (start the measurement)

show status (show the status display)

3.2.3 DAq parameters

Most of the following parameters are automatically set by the software displayed in gray when the data acquisition starts:

Parameter 2 Time stamp for an event as obtained from the TDC in μ s. Setting this parameter to 1 or 2 will record the TDC clock with the event as 32bit or 64bit value from the time data acquisition start.

0 = no Timestamp,
1 = 32Bit Timestamp (Low.Low, Low.high)
2 = 64Bit Timestamp (Low.Low, Low.high, High.Low, High.high)

Parameter 5 Time scaling (internal parameter)
Used to calibrate the time stamp.

Parameter 6 DAq-version number (internal parameter)

Parameter 7 Start time of list mode file (internally set)

Parameter 8 DAq-ID (internal parameter)
DAq_ID_HPTDC 0x000008 for TDC8HP

Parameter 9 LMF-version number (internal parameter)

Parameter 20 TDC resolution in ns (internally set).
For the **TDC8HP** the resolution (LSB) is fixed to 25ps or 100ps.

Parameter 21 TDC data type information (internally set)
0 = Not defined
1 = Channel information
2 = Time information (in ns)

Parameter 32 number of channels to be read out

Parameter 33 maximum number of hits to be read out

Parameter 40 DataFormat (Internally set)

Parameter 50 Checknumber to test CCF, DAn and DAq for compatibility

Parameter 52 tdcmanager->TDCDriverVersion (Internally set)

Parameter 53 NumberOfDAQLoops (normally 1)
To increase online event reading speed you can increase this Value. If set to n then you'll process n Events in the DAq module before returning the last event to CoboldPC. In online processing only the nth event will be passed to the analysis but all events will be written to the LMF.
If set to 0, 1 is assumed.

3.2.4 Additional DAq parameters for TDC8HP

3.2.4.1 type - description

3.2.4.1.1 mask

a bit mask with each bit representing a channel. The mask has 3 areas.
0xaaabbbccc (hexadecimal representation)
aaa do represent the 3ed board channels from 1-8
bbb do represent the 2nd board channels from 1-8
ccc do represent the 1st board channels from 1-9
The least bit of a block represents channel 1

3.2.4.1.2 edge

0 = falling
1 = none
2 = rising

3.2.4.1.3 *int*

Standard integer value

3.2.4.1.4 *bool*

0 = false
1 = true

3.2.4.1.5 *time*

time given in ns (nano seconds). The input is of the c++ datatype double.

3.2.4.2 scope – description

3.2.4.2.1 *channel*

applies to the specified channel described in the mask

3.2.4.2.2 *global*

applies to all TDC8HP boards

3.2.4.2.3 *board*

applies only to a specified TDC8HP board. In **CoboldPC** board applies always to the 1st TDC8HP board.

3.2.4.3 Parameter - description

Parameter 60	Config File Flag 0 = Configuration over parameters 1 = Configuration over TDC8HPGlobal.cfg file Default = 0
Parameter 61	RisingEnable type = mask scope = channel min = 0x000000 max = 0x0ff0ff1ff default = 0x000000000 description: Record 0 to 1 transitions for these channels
Parameter 62	FallingEnable type = mask scope = channel min = 0x000000 max = 0x0ff0ff1ff default = 0x0000001ff description: Record 1 to 0 transitions for these channels
Parameter 63	TriggerEdge type = edge scope = global min = 0 max = 2 default = falling (0) description: Trigger on falling or rising edge
Parameter 64	TriggerChannel type = int scope = board min = 1 max = 9 default = 8 description: Which channel to use as a trigger

Parameter 65 OutputLevel
 type = bool
 scope = board
 min = 0
 max = 1
 default = 0
 description: Keep track of signal levels

Parameter 66 GroupingEnable
 type = bool
 scope = global
 min = 0
 max = 1
 default = 1
 description: Switches between different driver modes
 0 = 16ps bin size, range = ± 30 ms
 1 = 25ps bin size, range = ± 209 μ s

Parameter 67 AllowOverlap
 type = bool
 scope = global
 min = 0
 max = 1
 default = 0
 description: If a hit falls within multiple groups copy
 it to all groups or just to the first one

Parameter 68 TriggerDeadTime
 type = time
 scope = global
 min = 0
 max = 1s
 default = 310ns
 description: Time during which no new triggers are
 accepted after a trigger
 time in ns

Parameter 69 GroupRangeStart
 type = time
 scope = global
 min = -209000ms
 max = 209000ms
 default = -300ns
 description: Smallest time value relative to group marker
 to be included in group. (Absolute value, end of range
 for common stop
 time in ns

Parameter 70 GroupRangeEnd
 type = time
 scope = global
 min = -209000ms
 max = 209000ms
 default = 300ns
 description: Largest time value relative group marker to
 be included in group. (Absolute value, end of range for
 common stop
 time in ns

Parameter 71 ExternalClock
 type = bool
 scope = global
 min = false
 max = true
 default = false

description: Use external clock source
Please set to 1 if two TDCs are synced!

Parameter 72 OutputRollovers
 type = bool
 scope = global
 min = 0
 max = 1
 default = 0
 description: Include rollover markers in the data format.
 Rollovers can only be disabled if grouping is enabled.

Parameter 78 VHR
 Type = bool
 Scope = global
 min = 0
 max = 1
 default = 1
 description: 1 = TDC's bin size 25ps
 0 = TDC's bin size 100ps
 Include rollover markers in the data format.
 Rollovers can only be disabled if grouping
 is enabled.

Parameter 80 Flag to read INL (from TDC8HPINL.cfg)
 0 = do not process file
 1 = read file and set data
 default = 0
 note: 1 is not supported yet

Parameter 81 Flag to read DNLTapAdjust (from TDC8HPDLLTapAdjust.cfg)
 0 = do not process file
 1 = read file and set data
 default = 0
 note: 1 is not supported yet

Parameter 86 MMXEnable
 0 = disabled
 1 = enabled
 description: use MMX technology in the TDC driver.

Parameter 87 DMAEnable
 0 = disabled
 1 = enabled
 description: increases data transfer speed on PCI-bus.

Parameter 88 Time zero channel
 usually all TDC values are relative to the trigger
 signal. This means that the first signal in the trigger
 channel has by definition the value 0. If parameter 88 is
 greater than zero then the values in the trigger channel
 are replaced by the time difference to the last signal
 which occurred in the "Time zero channel" (even if this
 signal was outside the TDC range which is specified by
 parameters 69 and 70).
 0 = not used
 default = 0
 note: works only when parameter 66 = 0

Parameter 89 Trigger channel mask (bit mask)
 Enables additional trigger channels in addition to
 parameter 64.
 default = 0
 note: works only when parameter 66 = 0

3.2.5 DAq coordinates

According to the settings of these parameters above the **CoboldPC** program will retrieve the following coordinates and (if selected) will store them event by event to the hard disc.

The format is defined in the **CoboldPC** manual, each event is a n-tupel {...,...,...,...} of the consecutive coordinates as binary numbers depending on the settings of parameters 2, 32 and 33:

```
{
  T1Ch01n[, T1Ch01S01, ..., T1Ch01Sxn]   - xn = para 33
  ...., ...., ...., ...., ....,
  T1Chxm[, T1ChxmS01, ..., T1ChxmSxn]   - xm = para 32
}
```

Further coordinates are calculated by the DAn (data analysis part), however these will not be stored to disc but appended to the list, all coordinates (from DAq and DAn) are internally numbered:

```
pEventData[0]
pEventData[1]
pEventData[2]
. . .
```

For the "Startup.ccf" n is set to 1 (one hit read-out only) and m equals 4 (6 in case of the *Hexanode*, see additional manual), the number of stored DAq coordinates is 8 (12) if the timestamp is disabled, otherwise 10 (14).

3.2.6 DAn parameters and coordinates:

While the parameters of the DAq part have only the function to define and organize the hardware (and are mandatory), the DAn parameters are used in the data analysis part. The DAn DLL module is a data analysis subprogram that complements the raw DAq coordinates by computed coordinates, such as the position or time sum (TOF) derived from the raw data. It also comprises some correction, shifting and rotation computations and coordinate system transformations, so that the basic computations for experiments with a position and time sensitive detector are already available without changing the DAn DLL module supplied here.

The computations yield in an additional set of coordinates (DAn-coordinates) that are internally treated as independent coordinates and are internally listed by numbers, following the last hardware coordinate (although they are not stored to hard disc in the list-mode file). This DAn DLL module may be altered using a MS-C++ compiler (see **CoboldPC** help file) and the list of coordinates may be changed, creating additional coordinates (and parameters) for further computation, unused DAn coordinates may be removed. A newly defined coordinate is available for further computations. It is clear that the program will only operate well, if all definitions in the filename.ccf (e.g. the "Startup.ccf") are in accordance with the DAq and DAn DLL modules used. After the *new* or *start* command the program makes a consistency check and will eventually give an error message if the number of coordinates and parameters defined are not sufficient, however, it will not detect all eventual discrepancies.

Even though the parameters from 1 to 99 are mainly used for the DAq module some of this information is also useful for the data analysis. During offline analysis these parameters are automatically set from the parameter information (settings during data acquisition) that is stored in the lmf-file.

DAn-parameters used in the DAn-part can have the function of variables for computations, of pointers or of flags. Some are mandatory, some are optional. Standard DAn will use the parameter range 100-299.

For a detailed introduction into the DAn-part of the data treatment please refer to the **CoboldPC** help file.

3.2.7 Spectra and conditions

The final purpose of the data acquisition is to display and manipulate the acquired data. For this purpose it is possible to define *spectra* for display of all defined coordinates. A spectrum is a histogram with fixed bin width either with a one- or two dimensional array of bins. For a one-dimensional spectrum (for example a time spectrum) this array is a row along the ordinate (X-axis) of a graph, the bins to values of the corresponding coordinate. When data are acquired or re-sorted from a list-mode file, value of the coordinate for each event will be attributed to the closest bin's value and the histogram content will be incremented by one unit (along the Y-direction of the graph) in this bin. In the example such a graph would represent the probability of time differences as function of time for the investigated set of events.

Likewise it is possible to display two-dimensional spectra, i.e. the coincident occurrence of values in two coordinates within the corresponding bin widths (for example the 2d position distribution of the detected particles). To visualize such a histogram the two coordinates span a plane (X/Y), the value in each bin (Z) is displayed as gray or color code, or contour lines are used. The range of the displayed spectra in X, Y (and Z), the bin size and the “unit” of incrementing can be defined for optimal visualization and manipulation.

To analyze higher dimensional coordinate correlations it is possible to “gate” the sorting process into a histogram (spectrum) by defining a **condition**. Such a condition can be a “window” on the occurrence of a certain range of values in a third coordinate for the events. For example one needs to visualize the (2d) position spectra of particles as function of their time-of-flight (TOF). Then one can define several conditions (gates) on the TOF coordinate (e.g. time sum peaks) and several 2d position spectra with the different conditions. It is possible to link different conditions (e.g. by an “AND”) to allow the analysis of even higher dimensional coordinate correlations.

For details about the definition of spectra and conditions, for spectrum manipulation options and data I/O to other programs please refer to the **CoboldPC** manual. In the “Startup.ccf” you find some pre-defined conditions (as an example) and spectra that will allow you to view the most important coordinates. For example, you will immediately be able to see a position spectrum.

You may edit the “Startup.ccf” to adjust them to your needs, e.g. setting the right condition gates on the time sum peak(s) omitting spectra that you do not need, adjust parameters (for shifting or rotating the spectra, calibrating position and time), changing or adding spectrum definitions.

Please note that these functions are only “first level” modifications of the data acquisition and analysis option provided by **CoboldPC**. More advanced data treatments like defining new (computed) coordinates to the analysis can be done by additionally modifying the DAn DLL module using a MS C++ compiler. Please refer to the **CoboldPC** manual for details.

If you are ready to run a session with the hardware now, you may execute the “Startup.ccf” file again and click the hardware button. But before make sure to follow the steps in the “**Getting Started**” chapter in the “**Position and time sensitive multi-hit MCP delay-line detector system**” manual.

4 Know Bugs

The following behavior of **CoboldPC** software have been observed in the following cases

- Miss configured TDC8HP board (either by parameters or by TDC8HPglobal.cfg)
- Trigger signal missing or adapted to the wrong channel

CboldPC stops processing commands as “stop”, “pause”, “start” and “new”. Is one of these commands is given under the condition described above then **CoboldPC** stops processing all commands.

If CoboldPC starts to behave unexpectedly then follow the following instructions.

- Close the **CoboldPC** program (after about 5s there might appear a dialog box – Select the “FORCE” button)
- Open the task manager and verify that **CoboldPC** (CoboldPC.exe) is no longer in the task list. If it is still in the list, kill this process.

This behavior will be corrected in a later release.





List of Figures

FIGURE 1.1: TDC8HP PCI CARD	5
FIGURE 1.2: THE TDC8HP-PCI CARD. THE HIGH RESOLUTION CHANNELS 0-7 AND THE LOW RESOLUTION NIM-CHANNEL 8 ARE LOCATED ON THE LEFT. LOW RESOLUTION CHANNELS 9-20 ARE CONNECTED TO THE 26-PIN RIBBON CABLE CONNECTOR ON THE RIGHT SIDE. THE 10-PIN RIBBON CABLE CONNECTOR AT THE TOP IS USED TO CONNECT AND SYNCHRONIZE A TDC8PCI2-CARD TO THE TDC8HP-PCI.	6
FIGURE 1.3: DATA ACQUISITION IN TRIGGERED MODE. HITS WITHIN THE GROUP'S RANGE (GREEN) ARE ACQUIRED; HITS OUTSIDE THAT RANGE (RED) ARE REJECTED.	7
FIGURE 1.4: THE TREE DIFFERENT TYPES OF GROUPING DESCRIBED IN THE TEXT	8
FIGURE 1.5: THE CARD'S SERIAL NUMBER LOCATED ON THE PCI-INTERFACE CHIP. THE FIRST DIGIT DENOTES THE YEAR OF PRODUCTION MINUS 2000, THE SECOND IS A UNIQUE SERIAL NUMBER.....	12
FIGURE 1.6: THE DRIVER'S STATE MACHINE AS SEEN BY THE USER.....	17
FIGURE 2.1: EXTERNAL CLOCK MODULE	25
FIGURE 2.2: TWO TDC8HP CARDS CONNECTED WITH THE EXTERNAL CLOCK MODULE.	25

List of Tables

TABLE 1.1: FALLING TRANSITION	8
TABLE 1.2: RISING TRANSITION.....	8
TABLE 1.3: ERROR WITH CHANNEL INFORMATION.....	8
TABLE 1.4: ERROR NUMBERS.....	9
TABLE 1.5: GROUP	9
TABLE 1.6: ROLLOVER	9
TABLE 1.7: LEVEL INFORMATION	9
TABLE 1.8: RESOLUTION	9
TABLE 1.9: PARAMETER CONFIGURABLE BY THE USER.....	11
TABLE 1.10: DEPRECATED OR OBSOLETE PARAMETERS	11
TABLE 1.11: HIGH RESOLUTION INPUTS.....	20
TABLE 1.12: TRIGGER INPUT	20
TABLE 1.13: CLOCK INPUT	21
TABLE 1.14: LOW RESOLUTION INPUT	21
TABLE 1.15: TDC8 SYNC INPUT.....	21
TABLE 1.16: TIME BASE	22
TABLE 1.17: DATA RATE.....	22