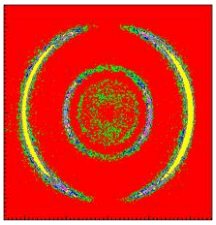
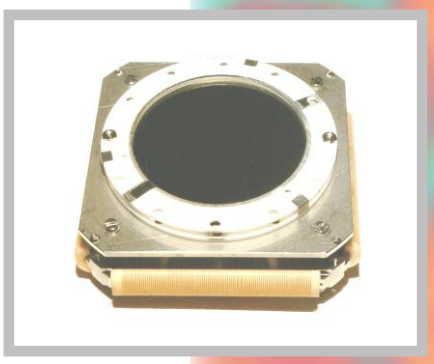
RoentDek
Handels GmbH

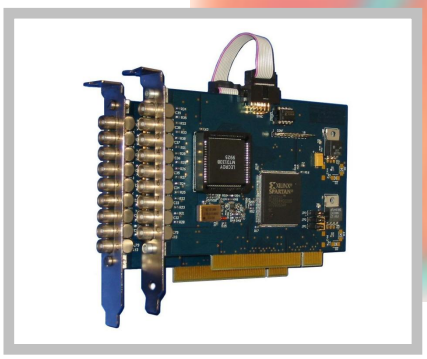
Supersonic Gas Jets
Detection Techniques
Data Acquisition Systems
Multifragment Imaging Systems

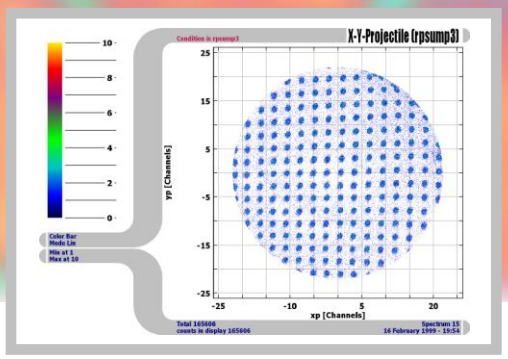


TDC8 ISA & PCI
and
TDC8PCI2
Manual

(9.8.907.1)







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

Table of Contents

1 The hardware

The **TDC8** is based on the LeCroy MTD133B chip. This card supports nearly all features of the MTD133B chip. Here the main features.

- Resolution 500ps/channel
- 16-Bit Dynamic Maximum Range
- Common Start and Common Stop (NIM input)
- 8 Channels (NIM input)
- Multi-Hit Operation, 1 to 16 hits, Programmable
- Double pulse resolution typically 10ns, guaranteed < 20ns
- ISA-Version: About 18k events/s/channel/hit on a 400MHz Pentium Computer running CoboldPC DAQ program. For a detector image with 4 channels and 1 hit/channel the event-rate can be up to about 5000 events/s
  PCI-Version: About 30k events/s on a 400MHz Pentium Computer running CoboldPC DAQ program. For a detector image with 4 channels and 1 hit/channel the event-rate can be up to about 25000 events/s

1.1 ISA-Bus TDC8

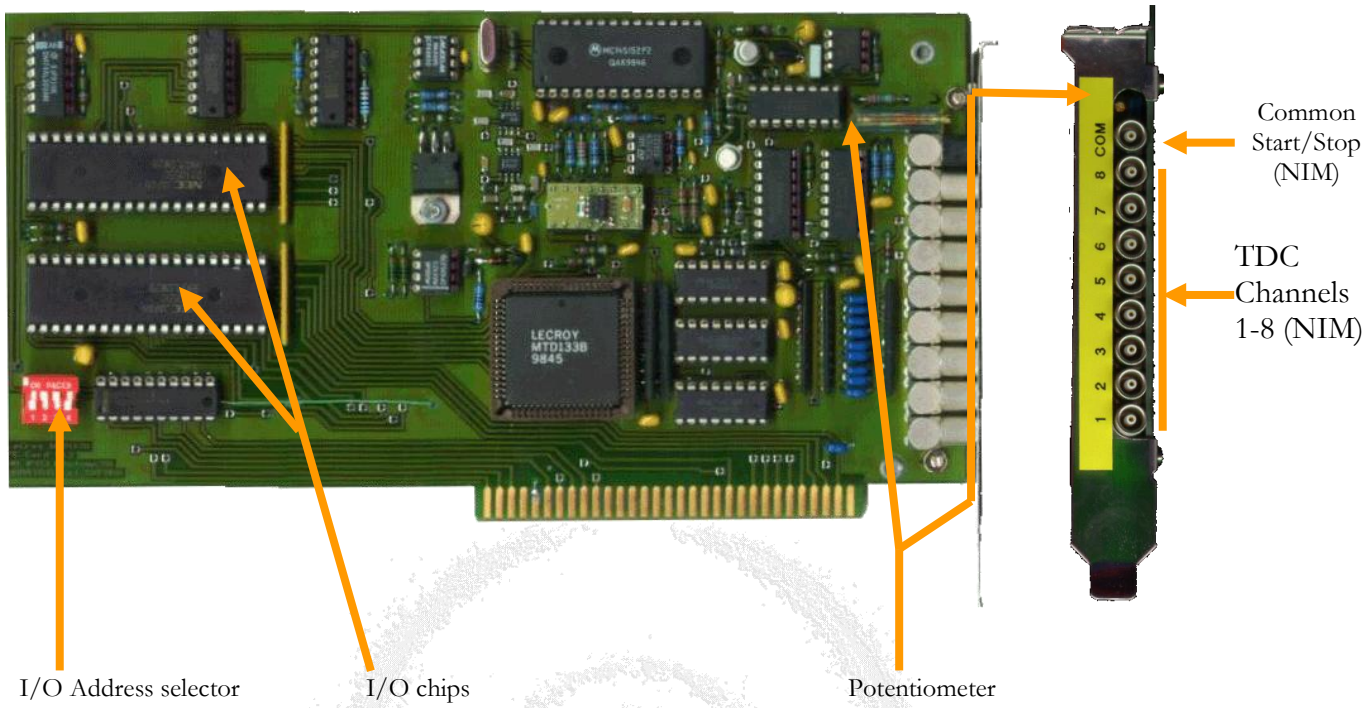


Figure 1.1: Front and side view of the TDC8/ISA board.

Never ever modify the settings of the potentiometer, unless you have a TDC8ISA with additional open time monitor output (see chapter 4.2 and Figure 1.2: TDC8PCI board and front view)!

With the I/O address selector you can select the following I/O addresses (hexadecimal values).

	dip switch number			
Address	1	2	3	4
0x300	On	On	On	Off
0x320 (default)	Off	On	On	Off
0x380	On	On	Off	Off

Table 1: TDC8 ISA Dip-Switches for I/O selection

1.2 PCI-Bus TDC8

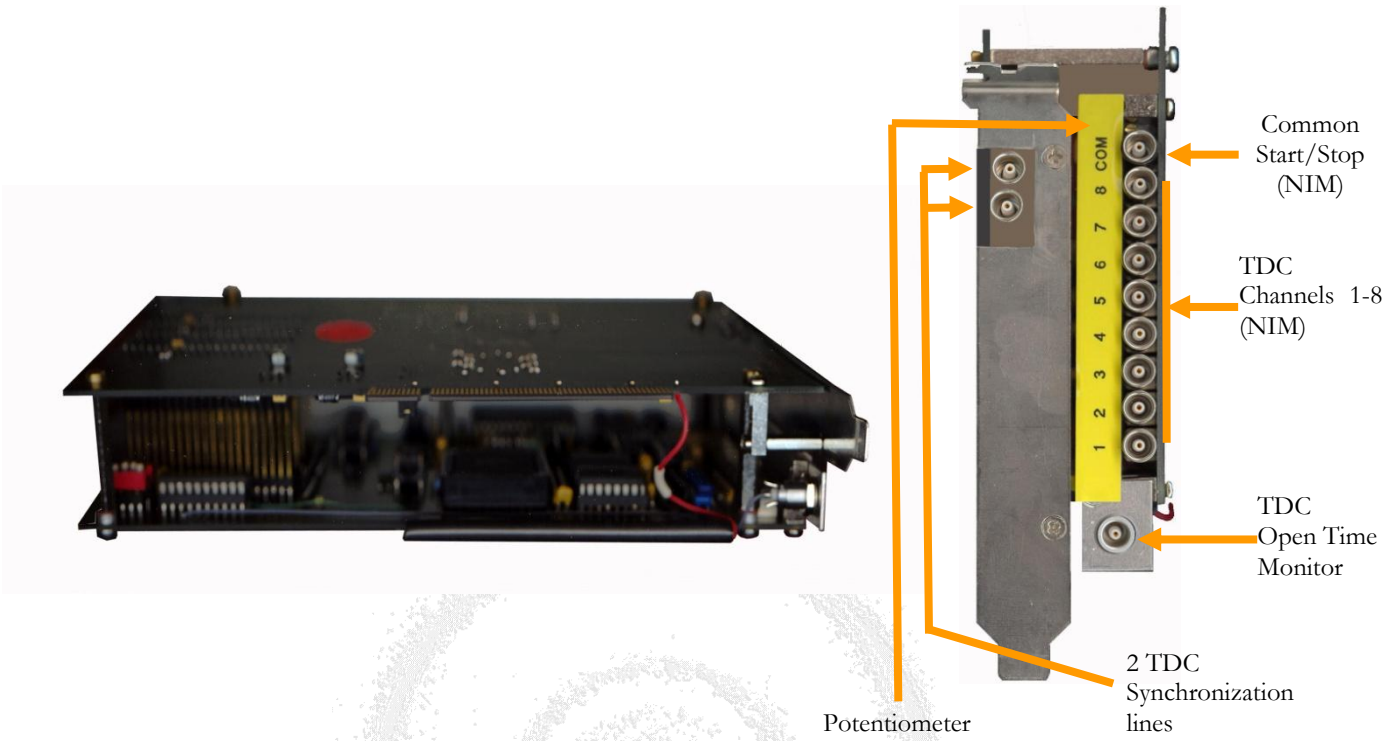


Figure 1.2: TDC8PCI board and front view

1.3 TDC8PCI2

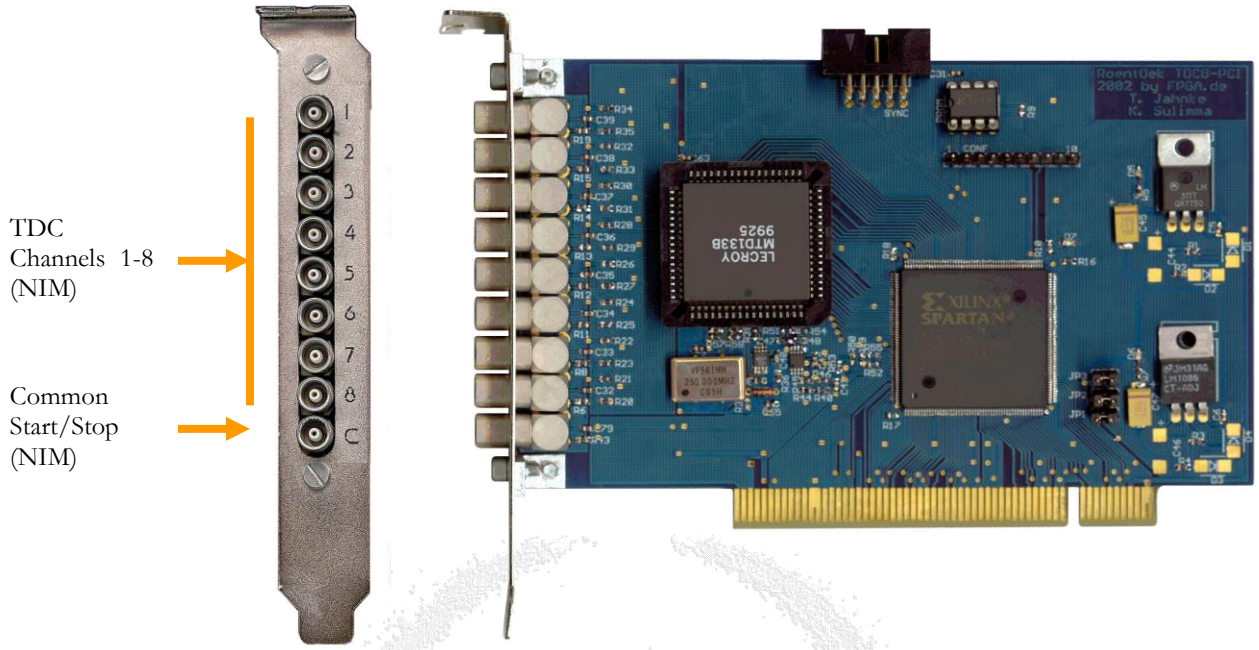


Figure 1.3: TDC8PCI2 board and side view

1.4 Two TDC8PCI2 synchronized

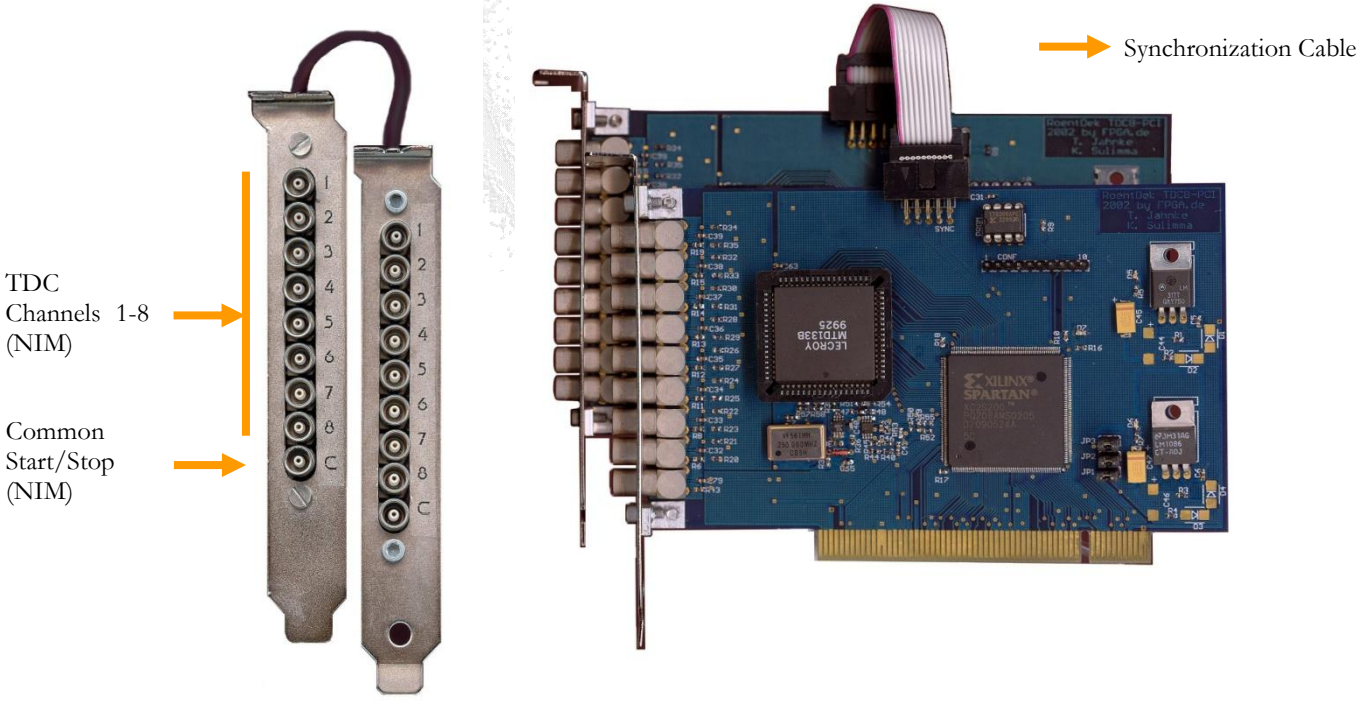


Figure 1.4: Two TDC8PCI2 in synchronized configuration

1.5 What is new with the TDC8PCI2

- First of all the **TDC8PCI2** is a single board TDC and not a back packed board like the **TDC8PCI.** Therefore it used only one PCI slot.
- The Gate Open time is now software adjustable. The Gate Open Monitor (NIM connector) is no longer available. Even though the actual value can be obtained by reading a certain configuration register.
- Two modules are now internally synchronized by a flat ribbon cable between the two modules.
- A Gate delay can now be programmed for Common Start Mode operation. This mode can be used to block events coming during this "Gate Delay" time.
- The DeviceID has changed from 0x2001 to 0x2004
- It is now possible to choose the TDC triggers. The rising and/or falling edge of the signal is selectable.

2 Installation of the TDC8 ISA or PCI

- 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.
- Adjust the I/O address setting on the card to a free I/O address.
  Do not forget to adjust parameter 1 in your .ccf file to this I/O address. For the PCI-Version set this parameter to 0.
- Locate a free ISA or 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.
  The **TDC8PCI** needs two PCI slots even though it connects only to one PCI slot connector.
  The **TDC8PCI2** needs only one PCI slot!
- 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.

Note for TDC8PCI(2) board!
Normally the PCI support in the BIOS is set to "Plug and Play" for operating systems that can handle plug and play components like Windows 2000 or Windows XP. In very rare occasions, the TDC is not working in this mode. In this special case the TDC card is detected but no data taking can be initiated. A DAQ Software like CoboldPC will therefore give no warning that the TDC could not be detected but the event rate will always be zero.
In this case try to switch the PCI support in BIOS from "Plug and Play" to "None Plug and Play" and try again.

3 Firmware Update

3.1 TDC8PCI

Please follow the instructions:

- Turn your computer off and unplug the power!
- Dismount the TDC8PCI card
- Locate the chip and replace it with the new one
  Please verify the correct marking on the chip and its position on the PCI card!

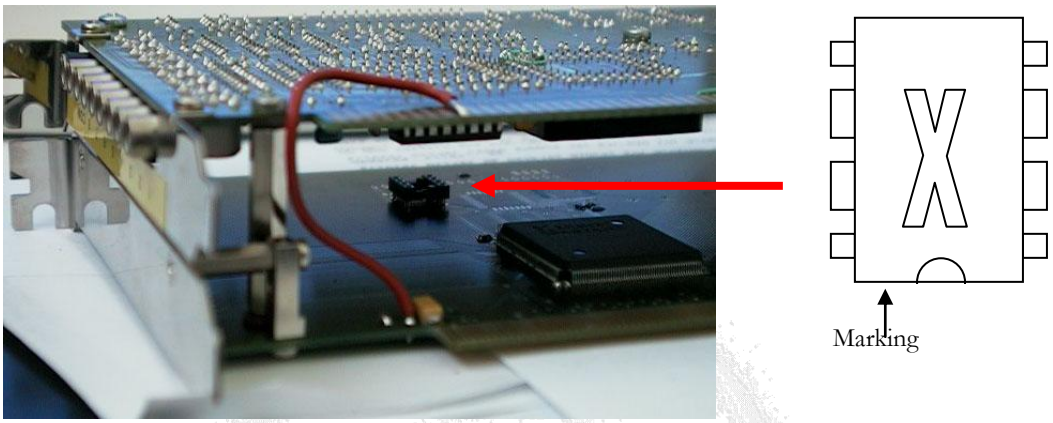


Figure 3.1: Firmware Chip replacement for TDC8PCI

Then mount the card again in your PC… Finished.

3.2 TDC8PCI2

Please follow the instructions:

- Turn your computer off and unplug the power!
- Dismount the TDC8PCI2 card
- Locate the chip and replace it with the new one
  Please verify the correct marking on the chip and its position on the PCI card!

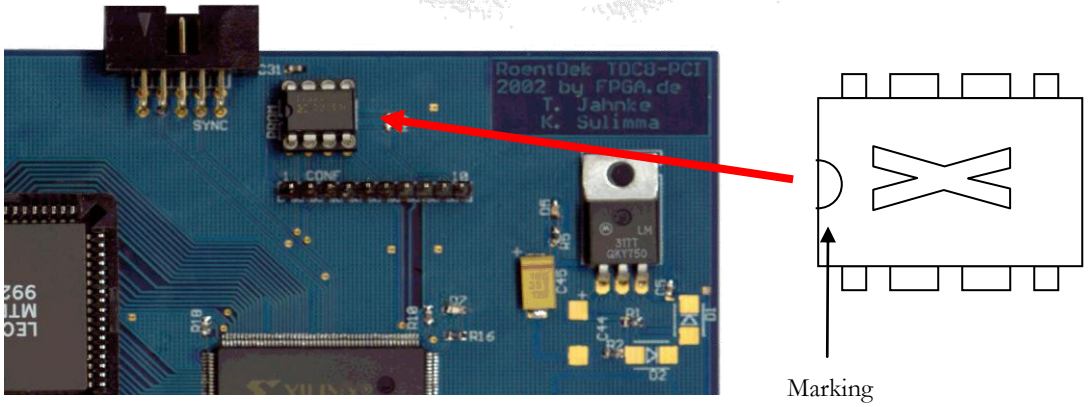


Figure 3.2: Firmware Chip replacement for TDC8PCI2

Then mount the card again in your PC… Finished.

4 Using the TDC8 card

4.1 General operation information for the TDC8 (ISA and PCI)

Typically the **TDC8** is to be operated as a single module. The **TDC8PCI** can be easily operated in two module mode. For the **TDC8ISA** this is, due to hardware restrictions on the **TDC8** and the PC hardware, not as easy as with the PCI version.

4.1.1 Two module mode with TDC8ISA

If you have to operate two **TDC8ISA** modules in one PC the **TDC8ISA** cards have to have different IO addresses!!! Please verify your hardware settings. By default the IO address is set to (hex) 320. Do not forget to adjust also the parameters for the IO address in your CoboldPC ccf file.
A typical problem in this configuration is to synchronize to two ISA boards. To access the **TDC8ISA** board you have to read out first one board, then the other. After reading the data from a **TDC8ISA** board the board is immediately ready to take new data. The time difference between the readout of the first and the second **TDC8ISA** board is several 10µs depending on the amount of data to be transferred. We suggest that you split one channel of the one board and apply the data simultaneously to the second **TDC8ISA** board. (In fact you are loosing one channel on one board). During data analysis you have to process only events that have the (nearly) same data for that splitted signal.

4.1.2 Two module mode with TDC8PCI

Because PCI is assigning the IO address automatically you only have to insert the PCI cards to free slots in your PC. Now connect the two synchronization lines diagonally as shown in Figure 4.1.

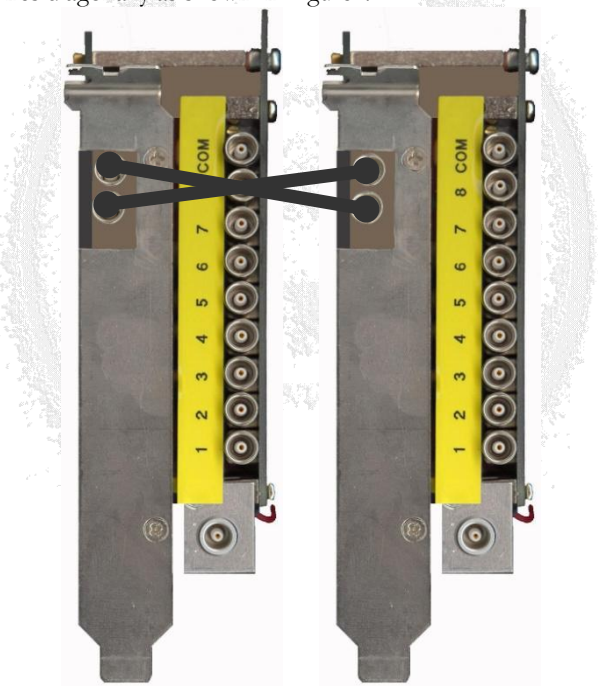


Figure 4.1: Connecting two TDC8PCI

All eight channels on each board can be used. The synchronisation is performed automatically by the PCI hardware. A sample output from the TDC Open Time output (monitor) is shown in Figure 4.3.

4.1.3 Two module mode with TDC8PCI2

Because PCI is assigning the IO address automatically you only have to insert the PCI cards to free slots in your PC. Now connect the synchronization cable as shown in Figure 4.2.

All eight channels on each board can be used. The synchronisation is performed automatically by the PCI hardware. With this card there is no TDC Open Time output (monitor) available.

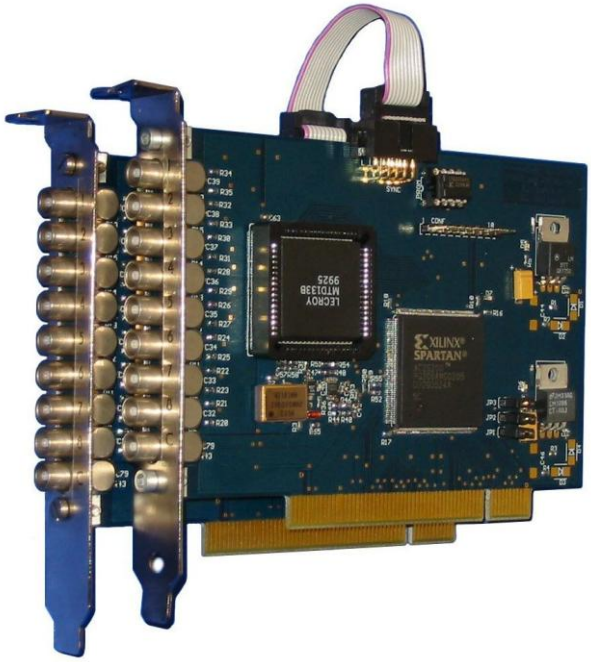


Figure 4.2: Connecting two TDC8PCI2

4.2 Adjusting the TDC Open Time

Where available on the **TDC8** board the TDC open time can be adjusted easily. By default the open time is adjusted to the 32µs measure range of the **TDC8**.
To adjust the open time you have to apply data to the COM input and to one of the eight channels. Then start the **TDC8PCI** in "Common Start" mode (for example by using CoboldPC). While the TDC is taking data the open time can be adjusted by using the potentiometer. The open actual open time can be monitored by the "TDC Open Time Monitor" output signal.

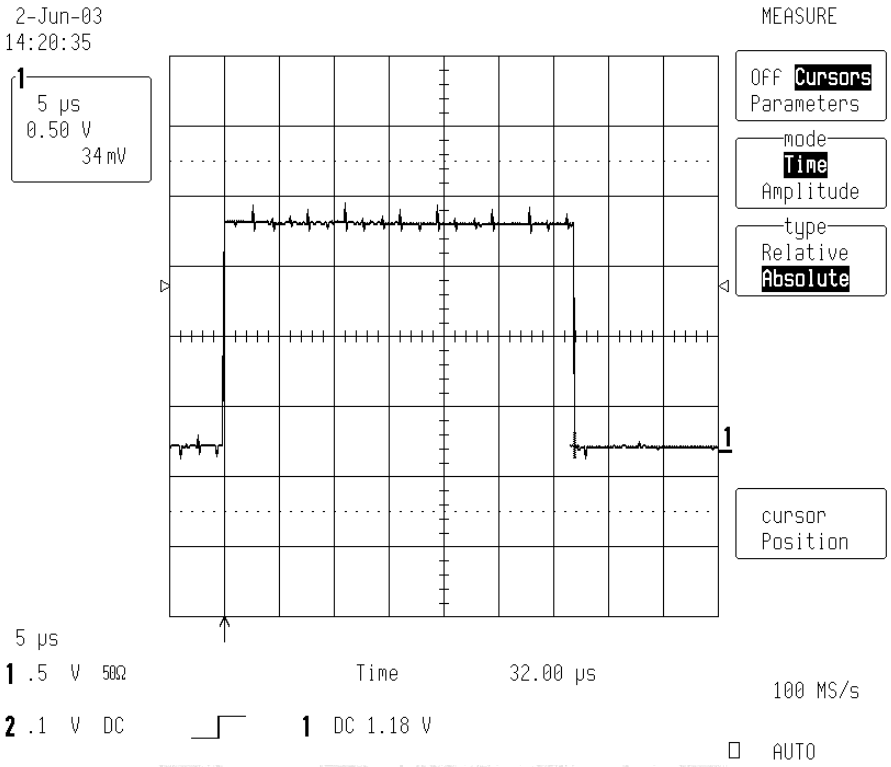


Figure 4.3: TDC8 Open Time adjustment (Monitor Output)

4.3 General information to program the TDC8/ISA

To handle all operations of the **TDC8/ISA** card you have to access 6 I/O port addresses. The I/O Base Address is set via the DIP-Switches (see. Table 1)

PIA1PA = I/O Base Address + 0 (input/output)
PIA2PA = I/O Base Address + 4 (input/output)
PIA1PB = I/O Base Address + 1 (input)
PIA2PB = I/O Base Address + 5 (output)
Control1 = I/O Base Address + 3 (input/output)
Control2 = I/O Base Address + 7 (input/output)

Control1 and Control2 are the configuration register of the 8255 I/O chips.

PIA1PA is input/output high byte data
PIA2PA is input/output low byte data
PIA1PB-Bit1 = p.out*
PIA1PB-Bit2 = channel number bit 0
PIA1PB-Bit3 = channel number bit 1
PIA1PB-Bit4 = channel number bit 2
PIA1PB-Bit7 = COM Disabled (Data collection finished)
PIA2PB-Bit0 = Enable*

PIA2PB-Bit3 = RESET
PIA2PB-Bit4 = p.in*
PIA2PB-Bit5 = Com-operation Mode (HI = common start)
PIA2PB-Bit6 = Common Stop trigger _/‾
PIA2PB-Bit7 = RCLK

4.4 Operational description

4.4.1 Configuring the 8255 I/O chips

- Set Control1 to hex 0xc3 and Control2 to hex 0xc1

Configuring the MTD133B chip

- Set Mode bit with Enable* and p.in* in PIA2PB
- Write ConfigHighByte to PIA1PA (setup number of Hits bits 0-3 (0 = 16 Hits))
- Write ConfigLowByte to PIA2PA (setup max wait time 8ns + bits(4-15) * 0.5ns)
- Reset TDC chip in Common Start Mode writing hex 0x39 and then 0x30 to PIA2PB
- Reset TDC chip in Common Stop Mode writing hex 0x19, 0x51 and 0x10 to PIA2PB

4.4.2 Test for present data

- Read PIA1PB and wait till Bit 7 is set.

4.4.3 MTD133B acquisition and readout

- Send 3 pulses to PCLK by toggling the PIA2PB Bit 7. (Enable, p.in* and Mode set, RCLK toggle)
- Then disable p.in* and get the Status. If PIA1PB Bit 1 (p.out) is set data is ready for transfer.
- Toggle RCLK line one more time.
- Get TDC data by reading PIA2PA for lowByte and PIA1PA for highByte.
- Get Status again to extract the channel number by reading PIA1PB.
- Store channel number and data value of the channel. Increment the hit counter of this channel.
- If bit 2 (p.out*) is still set then there is more data available so loop to get the data.
- After all data is read reenable PIA2PB bit 4 (p.in*).

4.4.4 C Sample Source Code for TDC8-ISA initialization and read-out

The source is not ready for compilation. It is just a fragmentational example of how access the **TDC8ISA** card.

```
#include <GeneralIO.h>                          // see sample CoboldPC sources
#include <HighResolutionTimer.h>                // see sample CoboldPC sources

int iPia1pa;
int iPia1pb;
int iPia2pa;
int iPia2pb;
int iCtrl1;
int iCtrl2;
unsigned int iConfigHigh;
unsigned int iConfigLow;

int _iNumberOfHits;
int iEventOpenTime;
int iTriggerModeCommon;

int iEventOpenTime = 32;                        // in µs
double _dConfig = iEventOpenTime - 0.8 / 5e-4;
unsigned int _iConfig = (unsigned int)_dConfig;
int iTifferModeCommon = true;                   // set to common start

bool TDC8ISAInit()
{
    if(!OpenPortsForNT(true))                   // use GiveIO driver for port access in NT
        return false;
    // setting Addresses
    int iIoAdr = ISA_IO;
```

```
    iPia1pa = iIoAdr + 0;
    iPia1pb = iIoAdr + 1;
    iPia2pa = iIoAdr + 4;
    iPia2pb = iIoAdr + 5;
    iCtrl1  = iIoAdr + 3;
    iCtrl2  = iIoAdr + 7;
    // MTD133B initialization
    OutputPort(iCtrl1,0xc3);                          // setup 8285 IO chips
    OutputPort(iCtrl2,0xc1);
    iConfigLow = (_iConfig & 0x00f0) | (0);
    iConfigHigh = (_iConfig & 0xff00) >> 8;
    return true;
}

int ISAGetTDC()
{
    int ch;                           // local variable for channel number
    int delay;                        // local variable for data value of the TDC
    int aa;                           // local variable for status information of the TDC
    int iCount;                       // local variable for loops

    if(iTriggerModeCommon)            // select common start or stop mode
        OutputPort(iPia2pb,0x11);
    else
        OutputPort(iPia2pb,0x31);

    OutputPort(iPia1pa,iConfigHigh);  // setup number of Hits bits 0-3 (0 = 16 Hits)
    OutputPort(iPia2pa,iConfigLow);   // setup max wait time 8ns + bits(4-15) * 0.5ns

    int WaitCount = 0;

    if(iTriggerModeCommon)            // Reset TDC
    {
        OutputPort(iPia2pb,0x19);
        OutputPort(iPia2pb,0x51);
        OutputPort(iPia2pb,0x10);
    }
    else
    {
        OutputPort(iPia2pb,0x39);
        OutputPort(iPia2pb,0x30);
    }
    do                 // wait for Event (after 100000 loops quit and start all over)
        aa = InputPort(iPia1pb);
    while(((aa & 0x80) != 0x80) && (WaitCount++ <= 100000));
    if(WaitCount >= 100000)           // if exit after 1000000 loops
        goto endget;                  // then signal no events.

    if(iTriggerModeCommon)            // set Enable* leave rest unchanged
        OutputPort(iPia2pb,0x11);
    else
        OutputPort(iPia2pb,0x31);

    for(iCount=1;iCount <= 3;iCount++)// 3 Pulses on RCLK*
    {
        if(iTriggerModeCommon)
        {
            OutputPort(iPia2pb,0x91);
            OutputPort(iPia2pb,0x11);
        }
        else
        {
            OutputPort(iPia2pb,0xb1);
            OutputPort(iPia2pb,0x31);
        }
    }

    if(iTriggerModeCommon)            // disable p.in*
        OutputPort(iPia2pb,0x01);
    else
        OutputPort(iPia2pb,0x21);

    aa = InputPort(iPia1pb);          // get status
```

```
    if((aa & 0x02))                    // and test for p.out*
    {
        memset(TDC,0,sizeof(TDC));     // clear all data of TDC array
        goto loop2;                    // if set then read channels
    }
    goto endget;                       // not set then quit -> no event

loop2:                                 // start of TDC readout
    if(iTriggerModeCommon)             // Pulse on RCLK*
    {
        OutputPort(iPia2pb,0x81);
        OutputPort(iPia2pb,0x01);
    }
    else
    {
        OutputPort(iPia2pb,0xa1);
        OutputPort(iPia2pb,0x21);
    }

    delay = InputPort(iPia2pa) | ((InputPort(iPia1pa)&0x7f) << 8);  // get TDC data
    aa = InputPort(iPia1pb);           // TDC status information
    ch = (aa & 0x1c) >> 2;             // select the channel number
    if((ch < MAXIMUM_NUMBER_OF_CHANNELS) &&
        (TDC[ch][0]++ +1 < MAXIMUM_NUMBER_OF_HITS+1))   // test valid hit # (hit count over increment)
        TDC[ch][TDC[ch][0]] = delay;   // if valid then transfer data to TDC array
    else
        goto loop2end;                 // if not valid then quit readout loop

    if(aa & 0x02)                              // test if more data is available
        goto loop2;                    // yes then loop2 again
loop2end:                              // no then follow up
    if(iTriggerModeCommon)             // reenable p.in*
    {
        OutputPort(iPia2pb,0x81);
        OutputPort(iPia2pb,0x11);
    }
    else
    {
        OutputPort(iPia2pb,0xa1);
        OutputPort(iPia2pb,0x31);
    }
    TDCHitsSum[TDC[0][0]] += 1;
    TDCHits[TDC[0][0]] += 1;

    return true;                       // signal good event

endget:
    return false;                      // signal bad/no event
}
```

4.5 General information to program the TDC8PCI(2)

The procedure described in chapter 4.4.1, 4.4.2 and 4.4.3 is here performed by an FPGA on the PCI adapter card.

To access the **TDC8PCI** you need especially two numbers:

```
Vendor          = 10dc (hex)
DeviceID        = 2001 (hex)
```

For the **TDC8PCI2** these numbers are:

```
Vendor          = 10dc (hex)
DeviceID        = 2004 (hex)
```

The technique used here is called "MemoryAccess".
RoentDek is using the MemAcc library* from ZeaL softstudios (http://zealsoft.com)
There you may download a version and documentation of their software.

Note: the LeCroy Chip has an undocumented "Feature". After receiving the Restart/Reset signal it has a dead time of about 5µs. So in Common-Start Mode a complete Cycle-Time is 32µs (Open time to collect Data) + 5µs Dead Time = 37µs. That means the theoretical maximum Event-Rate is about 27kHz!

4.5.1 Configuring the MemAcc library

Call the following functions:

```
maLicenseInfo("YourName",YourID);
maOpenLibrary();
```

4.5.2 Retrieve memory address of the TDC8PCI(2)

With

```
MyVendor = 0x10dc;
MyDeviceID = 0x2001 or 0x2004;
```

Call now

```
maGetDeviceBaseAddress(&MyVendor,&MyDeviceID,0,MyDevice)
```

to retrieve the base memory I/O address of the selected device
after that you have to map the physical (not accessible) memory address to your program memory by calling

```
maMapPhysToLinear(MyDevice[0].BaseAddress,MyDevice[0].Size, &MyDeviceHandle)
```

This function will return your program memory IO address for the selected device.

Now you access the **TDC8PCI(2)** only by this memory address.

* MemAcc library is © by ZeaL softstudios

4.5.3 TDC8PCI

Multiple cards sync-mode:
Multiple TDC8PCI cards can be used simultaneously if the needed number of TDC-channels exceeds 8. The sync-mode provides an event-based synchronization of several cards. The cards need to be connected via the sync connectors. By setting the "sync bit" in the "status register" the sync-mode is enabled.

4.5.3.1 Memory block used by the TDC8PCI

TDC8PCI:	**Vendor ID 0x10DC , Device ID 0x2001, Rev. 10 (X)**		
FIFO TOP (read top element of FIFO)			**[r]**
Address 0x0000 - 0x07FC:			
bit 31	**: FIFO empty**	**(1)**	
bit 30	**: toggle event bit**	**(0)**	
bits 26 – 24	**: channel**	**(0)**	
bits 19 – 16	**: 4 bit event counter**		
bits 15 – 00	**: time sample (in TDC bins)**	**(0)**	
FIFO GET (read and remove top element of FIFO)			**[r]**
Address 0x0800 - 0x0FFC:			
bit 31	**: FIFO empty**	**(1)**	
bit 30	**: toggle event bit**	**(0)**	
bits 26 – 24	**: channel**	**(0)**	
bits 19 – 16	**: 4 bit event counter**		
bits 15 – 00	**: time sample (in TDC bins)**	**(0)**	
Elements in FIFO (number of time samples currently stored in the FIFO)			**[r]**
Address 0x1000 - 0x17FC			
FIFO Size (maximum number of elements in FIFO)			**[r]**
Address 0x1800 - 0x1FFC:			
bits 31 – 00	**: size of FIFO [1025]**		
TDC Status			**[r/w]**
Address 0x2000 - 0x27FC:			
bit 31	**: FIFO empty**	**(1)**	**(write 1 to reset FIFO and readout)**
bit 30	**: FIFO full**	**(0)**	
bits 23 – 16	**: card revision number**		
bit 08	**: mode**	**(0)**	**0 = common stop, 1 = common start**
bit 02	**: write empty events**	**(0)**	**write 0 = discard empty events, 1 = empty events to FIFO**
bit 01	**: sync two cards**	**(0)**	**write 0 = no sync, 1 = do sync**
bit 00	**: TDC enable**	**(0)**	
TDC Range (number of time bins the TDC will accept)			**[r/w]**
Address 0x2800 - 0x2FFC :			
resets to 65535			
bits 31 – 16	**: are always 0**		
bits 03 – 00	**: are always 1**		
TDC Info			**[r/w]**
Address 0x3000 - 0x37FC :			
bits 31 – 24	**: number of channels**	**(8)**	
bits 23 – 16	**: bits per time sample**	**(16)**	
bits 15 – 08	**: max number of hits per channel**	**(16)**	
bits 07 – 00	**: number of hits per channel values larger than 16 are undefined**	**(1)**	
TDC Resolution (float) (size of a time bin in seconds)			**[r]**
Address 0x3800 - 0x3FFC :			
500e-12 <=> 500ps			
Read empty events counter			**[r]**
Address 0x4000 - 0x47FC :			
bits 31 – 00	**: number of empty events since last reset**		
Read and reset empty events counter			**[r]**
Address 0x4800 - 0x4FFC :			
bits 31 – 00	**: number of empty events since last reset**		

Table 2: Memory access table for TDC8PCI

4.5.3.2 C Sample Source Code for TDC8PCI initialization and read-out

The source is not ready for compilation. It is just a fragmentational example of how access the **TDC8PCI** card based on the memacc library

```
#include <HighResolutionTimer.h>
#include <MemAcc.h>

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

BADDR MyDevice[6];
USHORT MyVendor;
USHORT MyDeviceID;
unsigned int *iPCIIoAdr;
HANDLE MyDeviceHandle;

bool _bTDCFirstRead;
unsigned int _uiLastEventFirstFIFOData;
unsigned int _uiEventToggleFlag;
unsigned char *pucTDCChannel;        // channel at offset 2 from 32 bit information (INTEL FORMAT!)
unsigned int *puiFIFOGet;
unsigned int uiFIFOData;
unsigned short usTDCData;
unsigned char ucTDCChannel;
unsigned short TDC[MAXIMUM_NUMBER_OF_CHANNELS][MAXIMUM_NUMBER_OF_HITS+1];

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

int iEventOpenTime = 32;             // in µs
double _dConfig = iEventOpenTime - 0.8 / 5e-4;
unsigned int _iConfig = (unsigned int)_dConfig;
int iTifferModeCommon = true;        // set to common start

bool TDC8Init()
{
    // MemAccess Stuff
    maLicenseInfo("YourName",YourID);
    maOpenLibrary();
    MyVendor = 0x10dc;
    MyDeviceID = 0x2001;
    if(maGetDeviceBaseAddress(&MyVendor,&MyDeviceID,0,MyDevice) != 1)
    {
        AfxMessageBox("No TDC8/PCI could be found.");
        iPCIIoAdr = 0x0000;
        return false;
    }
    else
    {
        iPCIIoAdr = (unsigned int *)maMapPhysToLinear(MyDevice[0].BaseAddress,MyDevice[0].Size,
&MyDeviceHandle);
        if(!iPCIIoAdr)
        {
            AfxMessageBox("Can not map Physical to Linear Memory!");
            iPCIIoAdr = 0x0000;
            return false;
        }
    }
    iIoAdr = MyDevice[0].BaseAddress;

    pucTDCChannel = ((unsigned char *)&uiFIFOData)+3;       // channel at offset 2 from 32 bit
                                                            // information (INTEL FORMAT!)
    puiFIFOGet = &iPCIIoAdr[FIFOGET];
    ////////////////////////
    // TDC8 Stuff
    ////////////////////////
    // config, reset, enable
    iPCIIoAdr[TDCSTATUS] = 0x80000000;                      // reset and disable
```

```
    iPCIIoAdr[TDCINFO]    = 15;                                  // Get all hits... _iNumberOfHits;
                                                                 // info sets the #of Hits (1...16)
    iPCIIoAdr[TDCRANGE]   = iEventOpenTime*10000/5;              // µs*1e6/500
    iPCIIoAdr[TDCRANGE]   = 0xffff;                              // µs*1e6/500
    int iData = 0x00000001 | (((iTriggerModeCommon)&0x00000001) << 8);
    iPCIIoAdr[TDCSTATUS] = iData;                                // enable TDC and Common select
    _bTDCFirstRead = true;
    return true;
}

void TDC8Exit()
{
        maUnmapPhysicalMemory(MyDeviceHandle, iPCIIoAdr);
        maCloseLibrary();
}

int PCIGetTDC()
{
    int WaitCount = 0;
    unsigned int uiCEC = iPCIIoAdr[TDCCEC];
    if(_bTDCFirstRead)
    {
        do
        {
            // get the data
            uiFIFOData = *puiFIFOGet;
            if(!(uiFIFOData & 0x80000000))
            {
                 _bTDCFirstRead = false;
                break;
            }
            if(WaitCount++ >= 100000)
            {
                ////////////////////////
                // TDC8 Stuff
                ////////////////////////
                // config, reset, enable
                iPCIIoAdr[TDCSTATUS] = 0x80000000;               // reset and disable
                iPCIIoAdr[TDCINFO]    = 0;                       //_iNumberOfHits;
                                                                 // info sets the #of Hits (1...16)
                iPCIIoAdr[TDCRANGE]   = iEventOpenTime*10000/5;      // µs*1e6/500
                iPCIIoAdr[TDCSTATUS] = 0x00000001 | (((iTriggerModeCommon)&0x00000001) << 8);
                                                                 // enable TDC and Common select
                 _bTDCFirstRead = true;
                return false;
            }
        }
        while(true);
    }
    else
        uiFIFOData = _uiLastEventFirstFIFOData;

    int _iData = iPCIIoAdr[TDCSTATUS];
    memset(TDC,0,sizeof(TDC));                          // clear TDC array

    int _tCount = 0;

    // store the EventToggleFlag
    _uiEventToggleFlag = uiFIFOData & 0x40000000;    // prepare event toggle flag and store it

    // now process the data
    do
    {
        usTDCData = uiFIFOData & 0x0000ffff;
        ucTDCChannel = *pucTDCChannel & 0x1f;
        if(ucTDCChannel < MAXIMUM_NUMBER_OF_CHANNELS)
        {
            // increase Hit Counter;
            TDC[ucTDCChannel][0]++;

            // test for oversized Hits
            if(TDC[ucTDCChannel][0] > MAXIMUM_NUMBER_OF_HITS)
            {
                // # of hits > 16 then stop readout but signal good event
```

```
                // reset TDC and indicate no event

                ////////////////////////
                // TDC8 Stuff
                ////////////////////////
                // config, reset, enable
                iPCIIoAdr[TDCSTATUS] = 0x80000000;                  // reset and disable
                iPCIIoAdr[TDCINFO]   = 0;                           //_iNumberOfHits;
                                                                    // info sets the #of Hits (1...16)
                iPCIIoAdr[TDCRANGE]  = iEventOpenTime*10000/5;      // µs*1e6/500
                iPCIIoAdr[TDCSTATUS] = 0x00000001 | (((iTriggerModeCommon)&0x00000001) << 8);
                                                                    // enable TDC and Common select
                _bTDCFirstRead = true;
                return true;
            }
            // if Hit # ok then store it
            TDC[ucTDCChannel][TDC[ucTDCChannel][0]] = usTDCData;
        }
        uiFIFOData = *puiFIFOGet;                           // read new data from FIFO
    }
    while (!(uiFIFOData & 0x80000000) && ((uiFIFOData & 0x40000000) == _uiEventToggleFlag));
    // read as long as EventToggleFlag doesn't change

    // if running out of data then
    // signaling again first read of data!
    if(uiFIFOData & 0x80000000)
        _bTDCFirstRead = true;

    // store last FIFO data
    _uiLastEventFirstFIFOData = uiFIFOData;

    return true;
}
```

4.5.4 TDC8PCI2

The Gate Delay function is new for the **TDC8PCI2** module. All other functions are the same as for the **TDC8PCI** module.

Gate Delay:
[register value] * 30ns + 150ns = gatedelay[ns]
Values > 64 lead to minimal gate delay of 10ns

Gate Open Time:
[register value] * 30ns = gate opentime[ns]

Multiple cards sync-mode:
Multiple TDC8PCI2 cards can be used simultaneously if the needed number of TDC-channels exceeds 8. The sync-mode provides an event-based synchronization of several cards. The cards need to be connected via the sync connector. By setting the "sync bit" in the "status register" the sync-mode is enabled.

Note: Please make sure that the sum of GateDelay and GateOpenTime is smaller than the TDCRange. If the sum is greater and hits arrive between TDCRange and GateDelay+GateOpenTime then all data of that event will be lost and the hit counter is set to 0!

4.5.4.1 Memory block used by the TDC8PCI2

TDC8PCI:	**Vendor ID 0x10DC , Device ID 0x2004, Rev. 2 (0x02)**		
FIFO TOP (read top element of FIFO)			**[r]**
Address 0x0000 - 0x07FC:			
bit 31	**: FIFO empty**	**(1)**	
bit 30	**: toggle event bit**	**(0)**	
bit 29	**: empty event**		**valid event = 0, event invalid, did not contain data = 1**
bit 28	**: edge type**		**rising edge = 0, falling edge = 1**
bit 27	**: last data in event**	**(0)**	
bits 26 – 24	**: channel**	**(0)**	
bits 19 – 16	**: 4 bit event counter**		
bits 15 – 00	**: time sample (in TDC bins)**	**(0)**	
FIFO GET (read and remove top element of FIFO)			**[r]**
Address 0x0800 - 0x0FFC:			
bit 31	**: FIFO empty**	**(1)**	
bit 30	**: toggle event bit**	**(0)**	
bit 29	**: empty event**		**valid event = 0, event invalid, did not contain data = 1**
bit 28	**: edge type**		**rising edge = 0, falling edge = 1**
bit 27	**: last data in event**	**(0)**	
bits 26 – 24	**: channel**	**(0)**	
bits 19 – 16	**: 4 bit event counter**		
bits 15 – 00	**: time sample (in TDC bins)**	**(0)**	
Elements in FIFO (number of time samples currently stored in the FIFO)			**[r]**
Address 0x1000 - 0x17FC			
FIFO Size (maximum number of elements in FIFO)			**[r]**
Address 0x1800 - 0x1FFC:			
bits 31 – 00	**: size of FIFO [2049]**		
TDC Status			**[r/w]**
Address 0x2000 - 0x27FC:			
bit 31	**: FIFO empty**	**(1)**	**write 1 to reset FIFO and readout**
bit 30	**: FIFO full**	**(0)**	
bits 23 – 16	**: card revision number**		
bit 10	**: record rising edge**		**disabled = 0, enabled = 1**
bit 09	**: record falling edge**		**disabled = 0, enabled = 1**
bit 08	**: mode**	**(0)**	**0 = common stop, 1 = common start**
bit 02	**: write empty events**	**(0)**	**write 0 = discard empty events, 1 = empty events to FIFO**
bit 01	**: sync two cards**	**(0)**	**write 0 = no sync, 1 = do sync**
bit 00	**: TDC enable**	**(0)**	
TDC Range (number of time bins the TDC will accept)			**[r/w]**
Address 0x2800 - 0x2FFC :			
resets to 65535			
bits 31 – 16	**: are always 0**		
bits 3-0	**: are always 1**		
TDC Info			**[r/w]**
Address 0x3000 - 0x37FC :			
bits 31 – 24	**: number of channels**	**(8)**	
bits 23 – 16	**: bits per time sample**	**(16)**	
bits 15 – 08	**: max number of hits per channel**	**(16)**	
bits 07 – 00	**: number of hits per channel**	**(1)**	
	values larger than 16 are undefined		
TDC Resolution (float) (size of a time bin in seconds)			**[r]**
Address 0x3800 - 0x3FFC :			
500e-12 <=> 500ps			
Read empty events counter			**[r]**
Address 0x4000 - 0x47FC :			
bits 31 – 00	**: number of empty events since last reset**		
Read and reset empty events counter			**[r]**
Address 0x4800 - 0x4FFC :			
bits 31 – 00	**: number of empty events since last reset**		
Common-Start acquisition gate			**[r/w]**
Address 0x5000 - 0x57FC :			
bits 23 – 16	**: gate delay**	**(64)**	
bits 15 – 00	**: acquisition gate open time**	**(1080)**	

Table 3: Memory access table for TDC8PCI

4.5.4.2 C Sample Source Code for TDC8PCI2 initialization and read-out

The source is not ready for compilation. It is just a fragmentational example of how access the **TDC8PCI2** card based on the memacc library

```
#include <HighResolutionTimer.h>
#include <MemAcc.h>

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

BADDR MyDevice[6];
USHORT MyVendor;
USHORT MyDeviceID;
unsigned int *iPCIIoAdr;
HANDLE MyDeviceHandle;

bool _bTDCFirstRead;
unsigned int _uiLastEventFirstFIFOData;
unsigned int _uiEventToggleFlag;
unsigned char *pucTDCChannel;         // channel at offset 2 from 32 bit information (INTEL FORMAT!)
unsigned int *puiFIFOGet;
unsigned int uiFIFOData;
unsigned short usTDCData;
unsigned char ucTDCChannel;
unsigned short TDC[MAXIMUM_NUMBER_OF_CHANNELS][MAXIMUM_NUMBER_OF_HITS+1];

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

int iEventOpenTime = 32;              // in µs
double _dConfig = iEventOpenTime - 0.8 / 5e-4;
unsigned int _iConfig = (unsigned int)_dConfig;
int iTifferModeCommon = true;         // set to common start

bool TDC8Init()
{
    // MemAccess Stuff
    maLicenseInfo("YourName",YourID);
    maOpenLibrary();
    MyVendor = 0x10dc;
    MyDeviceID = 0x2004;
    if(maGetDeviceBaseAddress(&MyVendor,&MyDeviceID,0,MyDevice) != 1)
    {
        AfxMessageBox("No TDC8/PCI could be found.");
        iPCIIoAdr = 0x0000;
        return false;
    }
    else
    {
        iPCIIoAdr = (unsigned int *)maMapPhysToLinear(MyDevice[0].BaseAddress,MyDevice[0].Size,
&MyDeviceHandle);
        if(!iPCIIoAdr)
        {
            AfxMessageBox("Can not map Physical to Linear Memory!");
            iPCIIoAdr = 0x0000;
            return false;
        }
    }
    iIoAdr = MyDevice[0].BaseAddress;

    pucTDCChannel = ((unsigned char *)&uiFIFOData)+3;       // channel at offset 2 from 32 bit
                                                            // information (INTEL FORMAT!)
    puiFIFOGet = &iPCIIoAdr[FIFOGET];
    ////////////////////////
    // TDC8 Stuff
    ////////////////////////
    // config, reset, enable
    iPCIIoAdr[TDCSTATUS] = 0x00000000;                      // reset and disable
    iPCIIoAdr[TDCSTATUS] = 0x80000000;                      // reset and disable
```

```
    iPCIIoAdr[TDCINFO]    = 0x00;                                 // Get all hits... _iNumberOfHits;
                                                                  // info sets the #of Hits (1...16)
    iPCIIoAdr[TDCRANGE]   = iEventOpenTime*10000/5;               // µs*1e6/500
    iPCIIoAdr[TDCRANGE]   = 0xffff;                               // µs*1e6/500
    int iData =
        1 | // TDC enable
        (((bEmptyEvents)&0x00000001) << 2) |
        ((!iTriggerModeCommon)&0x00000001) << 8) |
        (((bFallingEdge)&0x00000001) << 10) |
        (((bRisingEdge)&0x00000001) << 9);
    iPCIIoAdr[TDCSTATUS] = iData;                                 // enable TDC and Common select
    _bTDCFirstRead = true;
    // Common start acquisition gate
    // bit 00 - 15 = gate open time  *30ns = gate open time[ns]
    // bit 16 - 23 = gate delay      *30ns+150ns = gate delay[ns]
    iPCIIoAdr[TDCCOMMONSTARTGATE] = iGateOpen + (iGateDelay << 16);
    return true;
}

void TDC8Exit()
{
        maUnmapPhysicalMemory(MyDeviceHandle, iPCIIoAdr);
        maCloseLibrary();
}

int PCIGetTDC(CDoubleArray *pParameter)
{
    int WaitCount = 0;
    if(_bTDCFirstRead)
    {
        do
        {
            // get the data
            uiFIFOData = *puiFIFOGet;
            if(!(uiFIFOData & 0x80000000))
            {
                 _bTDCFirstRead = false;
                break;
            }
            if(WaitCount++ >= 100000)
            {
                ////////////////////////
                // TDC8 Stuff
                ////////////////////////
                // config, reset, enable
                iPCIIoAdr[TDCSTATUS] = 0x00000000;                // reset and disable
                iPCIIoAdr[TDCSTATUS] = 0x80000000;                // reset and disable
                iPCIIoAdr[TDCINFO]   = 0;                         //_iNumberOfHits;
                                                                  // info sets the #of Hits (1...16)
                iPCIIoAdr[TDCRANGE]  = iEventOpenTime*10000/5;    // µs*1e6/500
                iPCIIoAdr[TDCSTATUS] =
                                    1 | // TDC enable
                                    (((bEmptyEvents)&0x00000001) << 2) |
                                    ((!iTriggerModeCommon)&0x00000001) << 8) |
                                    (((bFallingEdge)&0x00000001) << 10) |
                                    (((bRisingEdge)&0x00000001) << 9);
                 _bTDCFirstRead = true;
                return false;
            }
        }
        while(true);
    }
    else
        uiFIFOData = _uiLastEventFirstFIFOData;

    memset(TDC,0,sizeof(TDC));    // clear TDC array

    int _tCount = 0;
    // store the EventToggleFlag
    _uiEventToggleFlag = uiFIFOData & 0x40000000;    // prepare event toggle flag and store it

    // now process the data
    do
    {
```

```
        usTDCData = uiFIFOData & 0x0000ffff;
        ucTDCChannel = *pucTDCChannel & 0x07;
        if(ucTDCChannel < MAXIMUM_NUMBER_OF_CHANNELS)
        {
            // increase Hit Counter;
            TDC[ucTDCChannel][0]++;

            // test for oversized Hits
            if(TDC[ucTDCChannel][0] > MAXIMUM_NUMBER_OF_HITS)
            {
                // # of hits > 16 then there is something wrong
                // reset TDC and indicate no event

                ////////////////////////
                // TDC8 Stuff
                ////////////////////////
                // config, reset, enable
                iPCIIoAdr[TDCSTATUS] = 0x00000000;                  // reset and disable
                iPCIIoAdr[TDCSTATUS] = 0x80000000;                  // reset and disable
                iPCIIoAdr[TDCINFO]   = 0;                           //_iNumberOfHits;
                                                                    // info sets the #of Hits (1...16)
                iPCIIoAdr[TDCRANGE]  = iEventOpenTime*10000/5;      // µs*1e6/500
                iPCIIoAdr[TDCSTATUS] =
                                    1 | // TDC enable
                                    (((bEmptyEvents)&0x00000001) << 2) |
                                    ((!iTriggerModeCommon)&0x00000001) << 8) |
                                    (((bFallingEdge)&0x00000001) << 10) |
                                    (((bRisingEdge)&0x00000001) << 9);

                _bTDCFirstRead = true;
                return false;
            }
            // if Hit # ok then store it
            TDC[ucTDCChannel][TDC[ucTDCChannel][0]] = usTDCData;
        }
        uiFIFOData = *puiFIFOGet;     // read new data from FIFO
    }
    while (!(uiFIFOData & 0x80000000) && ((uiFIFOData & 0x40000000) == _uiEventToggleFlag));
    // read as long as EventToggleFlag doesn't change

    // if running out of data then
    // signaling again first read of data!
    if(uiFIFOData & 0x80000000)
        _bTDCFirstRead = true;

    // store last FIFO data
    _uiLastEventFirstFIFOData = uiFIFOData;

    if(TDC[1][1] == 0)
        return true;

    return true;
}
```

4.6 Using the TDC8 card with CoboldPC DAQ software

4.6.1 DAN and DAQ Modules

To operate your TDC8 modules please copy the appropriate DAN.dll and DAQ.dll files to the installation directory of CoboldPC. They are placed in the folder "DAN and DAQ\DotNet" that you find in the installation directory of CoboldPC.

TDC8	folder contains files for the TDC8 and TDC8PCI version
TDC8PCI2	folder contains files for the TDC8PCI2 version
2TDC8	folder contains files for double TDC8 and/or TDC8PCI version
2TDC8PCI2	folder contains files for double TDC8PCI2 version

The "2TDC8PCI2 Standard.ccf" is valid for double TDC8, TDC8PCI and TDC8PCI2 versions.

4.6.2 The "TDC8 Standard.ccf"

We have prepared the sample command file "TDC8 Standard.ccf" according to your DAQ hardware for first use. It provides already most of the desired data, i.e. 2d position spectra and time-of-flight spectra in various coordinate representations. The program calls several subprograms that define parameters and coordinates which are attributed to the data acquisition part and the data analysis part of the event handling. Finally it defines spectra (and conditions). Due to this modular construction it is possible to use almost the same data analysis sequences for different hardware (i.e. TDC types). Some users find this sequenced structure of the "TDC8 Standard.ccf" file not adequate for their work. If so you may create your own "TDC8 Standard_personal.ccf" by replacing the "exe" commands by directly pasting the subprogram commands into the new "TDC8 Standard_personal.ccf". Please observe the order of commands.
The standard defined coordinates, spectra and condition gates in the "TDC8 Standard.ccf" are (please refer also to the commented lines in the "TDC8 Standard.ccf"):

restart	(reset of earlier commands)
execute subDAQ\TDC8\Standard-Parameters.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Parameters.ccf	(executes the commands in the specific file)
execute subDAQ\TDC8\Standard-Coordinates.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Coordinates.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Spectra.ccf	(executes the commands in the specific file)
new	(defines the session type, calls selector box)
start	(start the measurement)
show stat	(show the status display)

4.6.3 The "TDC8PCI2 Standard.ccf"

We have prepared the sample command file "TDC8PCI2 Standard.ccf" according to your DAQ hardware for first use. It provides already most of the desired data, i.e. 2d position spectra and time-of-flight spectra in various coordinate representations. The program calls several subprograms that define parameters and coordinates which are attributed to the data acquisition part and the data analysis part of the event handling. Finally it defines spectra (and conditions). Due to this modular construction it is possible to use almost the same data analysis sequences for different hardware (i.e. TDC types). Some users find this sequenced structure of the "TDC8PCI2 Standard.ccf" file not adequate for their work. If so you may create your own "TDC8PCI2 Standard_personal.ccf" by replacing the "exe" commands by directly pasting the subprogram commands into the new "TDC8PCI2 Standard_personal.ccf". Please observe the order of commands.
The standard defined coordinates, spectra and condition gates in the "TDC8PCI2 Standard.ccf" are (please refer also to the commented lines in the "TDC8PCI2 Standard.ccf"):

restart	(reset of earlier commands)
execute subDAQ\TDC8PCI2\Standard-Parameters.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Parameters.ccf	(executes the commands in the specific file)
execute subDAQ\TDC8PCI2\Standard-Coordinates.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Coordinates.ccf	(executes the commands in the specific file)
execute subDAN\Standard-Spectra.ccf	(executes the commands in the specific file)

new (defines the session type, calls selector box)
start (start the measurement)
show stat (show the status display)

4.6.4 DAQ parameters

Parameter 1 Address of the I/0 card or 0 for PCI bus auto detection mode. PCI bus auto detection is only valid for the **TDC8** Modules with PCI IO cards. A value smaller than 16 specifies the Device # of the TDC8PCI(2) board starting with 0.

Parameter 2 Time stamp for an event as obtained from the PC in µs. Setting this parameter to 1 or 2 will record the computer clock with the event as 32bit or 64bit value from the time data acquisition start. Please note that the accuracy of the recorded time is not guaranteed.
The time information is also dependent on the mother board of your PC.

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

Parameter 3 System reset time (in seconds) in case of missing signals (do not change without consulting **RoentDek**)

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_RAW        0x000000    for RAW (no Data)
DAQ_ID_TDC8       0x000002    for TDC8/ISA/PCI
DAQ_ID_2TDC8      0x000005    for 2 TDC8
                               (Advanced Burst Mode)
```

Parameter 9 LMF-version number (internal parameter)

Parameter 14 Defines whether common start or common stop mode is used. Default is 0 = common start. If the common stop mode (= 1) is used, the TDC output values should be considered being negative numbers. In this case the definition of coordinates x1,x2,y1,y2,z1 and z2 as function of the raw TDC values CmH1 changes: the values are additionally multiplied by the factor (-1).

Parameter 20 TDC resolution (internally set).
For the **TDC8** the resolution (LSB) is fixed to 500ps, the range is 16bit (32µs).

Parameter 21 TDC data type information (internally set)

```
0 = Not defined
1 = Channel information
2 = Time information (in ns)
```

Parameter 30 Event open time in µs.
Maximal time after the start that the TDC waits for stops.

Parameter 32 number of channels to be read out

Parameter 33 maximum number of hits to be read out

Parameter 40 DataFormat (Internally set)

4.6.5 Additional DAQ parameters for TDC8PCI2

Parameter 45 gate delay

```
gate delay      *30ns+150ns = gate delay[ns]
```

Parameter 46 acquisition gate opentime

```
gate open time  *30ns = gate open time[ns]
```

Parameter 47 write empty events

```
0 discard empty events
```

```
                        1 write empty events
Parameter 48        trigger falling edge
                        0 disable
                        1 enable
Parameter 49        trigger rising edge
                        0 disable
                        1 enable
Parameter 50        EmptyCounter, sum (Internally set)
Parameter 51        EmptyCounter, since last Event (Internally set)
```

4.6.6 Additional DAQ parameters for two TDC8PCI or TDC8PCI2 Modules

Instead of using the "TDC8 Standard.ccf" or "TDC8PCI2 Standard.ccf" please use the "2TDC8PCI2 Standard.ccf" command file.

```
Parameter 34        2nd PCI card: number of channels to be read out
Parameter 35        2nd PCI card: maximum number of hits to be read out
Parameter 60        2 PCI Card mode
                        0 = Sync Test off
                        1 = Sync Test on
Parameter 61        2nd PCI card: Device # of the TDC8PCI(2) board
Parameter 65        gate delay (2nd TDC)
                        gate delay      *30ns+150ns = gate delay[ns]
Parameter 66        acquisition gate opentime (2nd TDC)
                        gate open time  *30ns = gate open time[ns]
Parameter 67        write empty events (2nd TDC)
                        0 discard empty events
                        1 write empty events
Parameter 68        trigger falling edge (2nd TDC)
                        0 disable
                        1 enable
Parameter 69        trigger rising edge (2nd TDC)
                        0 disable
                        1 enable
Parameter 70        EmptyCounter, sum (Internally set)
                         (since PCIVersion8 or PCI2)(PCI only) (TDC2)
Parameter 71        EmptyCounter, since last Event (Internally set)
                        (sincd PCIVersion8 or PCI2)(PCI only) (TDC2)
```

4.6.7 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 consutive coordinates as binary numbers depending on the settings of parameters 2, 32 and 33:

```
{
  TRaw1,TRaw2,TRaw3,TRaw4 - if selected   (TimeStamp raw information)
  S1,C1H1,...,C1Hn        - n = para 33   (H stands for hit number)
  ..., ..., ..., ..., ...,               (S stands for the status register)
  Sm,CmH1,...CmHn         - m = para 32   (C stands for TDC channel number)
}
```

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->GetAt(0)
pEventData->GetAt(1)
pEventData->GetAt(2)
. . .
```

For the "TDC8(PCI2) Standard.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 12 (16).
For the "2TDC8PCI2 Standard.ccf" n is set to 1 but m is set to 8 for both TDC modules.

4.6.8 DAN parameters and coordinates:

While the parameters 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 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 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 may be altered using a MS-C++ or DEC-Fortran compiler (see CoboldPC manual) and the list of coordinates my 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 "TDC8 Standard.ccf") are in accordance with the DAQ.dll and DAN.dll used. After the ***new*** or ***start*** command the program makes a consistency check and may give an error message if the number of coordinates and parameters defined are not sufficient, however, it will not detect all possible discrepancies.

4.6.8.1 DAN parameters

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 analyses. So some parameters are again listed here. During offline analysis these parameters are automatically set from the parameter information (settings during data acquisition) that is stored in the lmf-file. So these are DAN-parameters but they are reread from List-Mode file header.

```
Parameter  2      Save TimeStamp
                        0 = no Timestamp,
                        1 = 32Bit Timestamp      (Low.Low, Low.high)
                        2 = 64Bit Timestamp      (Low.Low, Low.high, High.Low,
                                                  High.high)
Parameter  5      TimeScaling (Internally set, tics per s)
Parameter  6      DAQ Version # (Internally set)
Parameter  7      Start time of list mode file (internally set)
Parameter  8      DAQ_ID
                        DAQ_ID_RAW         0x000000    for RAW (no Data)
                        DAQ_ID_TDC8        0x000002    for TDC8/ISA/PCI
                        DAQ_ID_2TDC8       0x000005    for 2 TDC8
                                                        (Advanced Burst Mode)
Parameter 20      Resolution of TDC in ns (internally set)
                  For the TDC8 the resolution (LSB) is fixed to 500ps, the range
                  is 16bit (32µs).
Parameter 21      TDC data type information (internally set)
                        0 = Not defined
                        1 = Channel information
                        2 = Time information (in ns)
Parameter 32      number of Channels (reread during offline)
Parameter 33      number of hits (reread during offline)
Parameter 40      DataFormat (Internally set)
```

The following 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. The following parameters and coordinates are used in the standard "TDC8(PCI2) Standard.ccf":

```
Parameter 100     Conversion Parameter for RAW data
```

Usually (parameter value 0), the data output from a **TDC8** TDC channel is coded in *channel numbers*[*]. The *channel number* is the number of resolution bins (i.e. LSB). If it is set to 1 the unit is transformed to ns, using the TDC resolution value (parameter 20). If the parameter is 2, a position in mm is calculated, using the values of parameters 110 and 111 (and 112). The time sum values are in ns unless the parameter is 0.

Parameter 102 Hexanode calculations

0 = no Hexanode
1 = Hexanode

If a Hexanode is used additional calculations are required to retrieve the position information. For these parameters and coordinates please refer to the add-on manual.

Parameter 103 R-Phi conversion

0 = RAD [-π..π]
1 = RAD [0..2π]
2 = DEG [-180..180]
3 = DEG [0..360]

This parameter defines the angular range and unit for the Phi coordinate in the R-Phi representation of the 2d-image.

Parameter 105 Start of DAQ Data for DAN

This pointer value defines for the DAN program part the position in the coordinate list where the first of the TDC data appears (s1). Usually you can set this value also to 0 and the program will automatically enter the right number.

Parameter 106 Start of DAN Data

This pointer value defines the position in the coordinate list where the DAN coordinates begin, i.e. it should equal the number of hardware coordinates
(See chapter 4.6.7)
If you want to analyze the data from the first hit you can set this value also to 0 and the program will automatically enter the right number.

Parameter 107 Hit number to be analyzed. Usually the position is calculated from the first hit in the TDC channels (default value: 1). If you want to get position and time sum calculations with the standard "TDC8 Standard.ccf" for a different hit number you have enter the hit value here. Note, that it can happen that the registered *channel numbers* do not necessarily correspond to the real particle hit if reflections on the raw amplifier signals produce "false" additional hits in a certain TDC channel number, or if hits are "lost" due to low signal height/high threshold settings.

Parameter 110 pTPCalX

Time to Position calibration factor for x ($v_\perp$ in mm/ns)
DLD40: 1.32, DLD80: 1.02, DLD120: 0.77
For Hexanode* (u): HEX80: 0.737, HEX120: 0.583

Parameter 111 pTPCalY

Time to Position calibration factor for y ($v_\perp$ in mm/ns)
DLD40: 1.43, DLD80: 1.13, DLD120: 0.82
For Hexanode* (v): HEX80: 0.706, HEX120: 0.567

These two parameters define the value of position to time calibration, the effective signal propagation speed across the delay-line. It depends on the size and geometry of the delay-line used. The suggested values are only accurate within few percent for a given delay-line. If a higher precision is needed

[*] This expression is always written in *italic* font, not to be mistaken for the term "TDC channel", which denominates a TDC input slot.

one needs to make a position calibration with a test mask in front of the detector. If the detector shows an oval shape please exchange the values for X and Y (only for DLD) and try again to sort the data, eventually the physical dimensions of the anode have been exchanged during mounting.

Parameter 112 pTPCalW
Time to Position calibration factor for Hexanode* (w):
HEX80: 0.684, HEX120: 0.540

Parameter 120 pCOx Rotation Offset Center for PosX

Parameter 121 pCOy Rotation Offset Center for PosY

These parameters define the center point for an online detector image rotation and also the center point in the X/Y plane for a coordinate transformation into R/Phi representation. Note that a R/Phi transformation will only give good results if the position unit is mm (see parameter 100).

Parameter 122 pRotA Rotation Angle mathematical direction
Rotation angle (counter clock wise) for an online detector image rotation
(value to be supplied in RAD or DEG depending on parameter 103)

Parameter 125 TDC channel number p of the MCP signal (default 0). If the MCP timing signal is not used for the common start or common stop, the x1,x2,y1,y2,z1 and z2 coordinate definitions are modified: The raw values CmH1 are reduced by the raw value in the TDC channel p (CpH1) of the MCP signal in this TDC channel number p before further computation according to parameters 14 and 21 are eventually performed. p = 0 means: no substraction. The MCP timing signals must then be connected to TDC channel p.

Parameter 135 pOPx Offset for PosX

Parameter 136 pOPy Offset for PosY

These two parameters are offset (additive) constants for shifting the detector image in the X/Y plane. Note, that in case of the Hexanode these values define the offsets for the calculated x and y and not for the raw u and v values.

Parameter 137 pOPw
Offset for third anode layer (added to w, only for Hexanode)

Parameter 138 pOSum
Offset for Sum/Diff calculations
This offset value is an additive constant to all time sum/diff coordinates

4.6.8.2 DAN coordinates, primary

The DAN coordinates are by definition only the additional coordinates that are computed from the (raw) DAQ coordinates retrieved from the hardware or from a previously accumulated event file. This "TDC8(PCI2) Standard.ccf" picks only one set of delay-line coordinates for one of the hits (default: first hit, see parameter 105) and calculates position and time values for these coordinates. If you have changed parameter 2, 32 or 33 from their default value (first hit only) or if you sort a list-mode file acquired with a non-default parameter settings) you need to adjust the (pointer) parameters 105 and 106. It is such possible to apply the position and time calculations to the next hits if such are (or have been) acquired by adjusting these pointer parameters. The DAN.dll will read the values of the status registers and the *channel numbers* in the 4 (Hexanode: 6) coordinates defined by parameter 105 (default: first hits) and calculate the desired position and time informations. Note that even for the use of a DLD (4 delay-line signals only), the coordinates for two additional delay-line signals (as from the Hexanodes) are defined and set to 0. A first set of DAN coordinates is created by using the defined set of DAQ coordinates:

AbsoluteEventTime absolute time of event from the start of data acquisition in µs (only if enabled, see parameter 2)

DeltaEventTime time between an event and the previous event in µs (only if time stamp recording is enabled, see parameter 2). This spectrum can be used to determine the average event

* please note that it is required to calibrate these numbers for your anode more accurately. Please contact service@roentdek.com.

	rate (use the "fit exp" CoboldPC command on the acquired spectrum)
EventCounter	number of event from the start of data acquisition
True	internal coordinate
ConsistenceIndicator	The value of this number for each event is: $\sum u \cdot 2^{i-1}$, i is the TDC channel, u =1, if at least one hit in the TDC channel i was registered, otherwise 0. If each TDC-channel for the selected hit number has received at least one hit of the value is 15 for a DLD and 63 for a Hexanode. This assumes that the first TDC channels are used for the delay-line signals. Up to 16 TDC channels are supported by this function.
PLLstatus	not used for **TDC8** (but must be defined)
n1	number of hits in TDC channel 1
n2	number of hits in TDC channel 2
n3	number of hits in TDC channel 3
n4	number of hits in TDC channel 4
n5	number of hits in TDC channel 5
n6	number of hits in TDC channel 6
x1	*channel number* of hit in channel 1 (default: hit 1)
x2	*channel number* of hit in channel 2 (default: hit 1)
y1	*channel number* of hit in channel 3 (default: hit 1)
y2	*channel number* of hit in channel 4 (default: hit 1)
z1	*channel number* of hit in channel 5 (default: hit 1) only for Hexanode
z2	*channel number* of hit in channel 6 (default: hit 1) only for Hexanode
	The values in these coordinates are calculated from the retrieved *channel numbers* of the selected DAQ-coordinates, e.g. (see above). Depending on parameters 100, 101, 104 these values have the specified units (corrected or uncorrected) and are the basis for all following computations. If a Hexanode is not used, z1 and z2 are set to zero. Note that channel 5 and 6 (for **TDC8**) can still be used for other timing signals. The corresponding coordinates are the DAQ coordinates for these TDC channels

These DAN coordinates are called primary because they retrieve the basic information in the DAQ coordinates for a first data review, assuming a delay-line detector is used. The following secondary DAN coordinates are computed from the primary coordinates and represent the first step of a (user defined) more elaborated data analysis. If you want to define additional coordinates you should append them to the secondary DAN coordinates. Here, basically the position in a given direction (e.g. x = x1 - x2) and the time sums (e.g. sumx = x1 + x2) are calculated from the primary DAN coordinates. Note that the "unit" of the secondary DAN coordinates is also defined by parameter 100. Additional shift parameters can be included and coordinate transformation or image rotation codes are provided. For the Hexanode please refer to the add-on manual.

4.6.8.3 DAN coordinates, secondary

x	x coordinate of the event	(x = x1 – x2)
y	y coordinate of the event	(y = y1 – y2)
w	set to zero	
sumx	time sum of x	(sumx = x1 +x2 + pOSum)
sumy	time sum of y	(sumy = y1 +y2 + pOSum)
sumw	set to zero	
sumxyw	sum of time sums	(sumxyw = sumx + sumy - pOSum)
diffxy	difference of sums	(diffxy = sumx – sumy + pOSum)
PosX	x-position	(PosX = x + pOPx)

```
PosY              y-position                    (PosY = y + pOPy,
                                                      If hex flag not set)
                                                (PosY = Yuv,
                                                      If hex flag set)
r                 r coordinate after transformation in r/phi coordinates
                  (from PosX/PosY)
phi               phi coordinate after transformation in r/phi coordinates
                  (from PosX/PosY)
xRot              x-position after rotation
yRot              y-position after rotation

The following coordinates are only filled with valid information for the Hexanode
setup. Even though they have to be defined!

Xuv               x + pOPx
Yuv               1/sqrt(3) * (x-2y) + pOPy
Xuw               Xuv
Yuw               1/sqrt(3) * (2w-x) + pOPy
Xvw               (y+w) + pOPx
Yvw               1/sqrt(3) * (w-y) + pOPy
dX                Xuv - Xvw
dY                Yuv – Yvw
```

4.6.9 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 "slots". For a one-dimensional spectrum (for example a time spectrum) this array is a row along the ordinate (X-axis) of a graph, the slots (or 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 "subDAN\Standard-Spectra.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 now edit the "TDC8(PCI2) Standard.ccf" and all subprograms (especially the "subDAN\Standard-Spectra.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 using a MS C++ or FORTRAN compiler. Please refer to the **CoboldPC** manual for details.
Also, there is a "zero-level" of operating for the recent **CoboldPC 2002** versions, allowing addressing the **CoboldPC** commands by a scripting language. Again, for details refer to the **CoboldPC** manual.

If you are ready to run a session with the hardware now, you may execute the "TDC8(PCI2) Standard.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.

List of Figures

List of Tables