cronologic

xTDC4-PCIe User Guide

www.cronologic.de

xTDC4-PCIe

Contents

1 Introduction

The xTDC4 is a *common-start* high resolution time-to-digital converter. Timestamps of leading or trailing edges of digital pulses are recorded. The xTDC4 produces a stream of output packets, each containing data from a single start event. The relative timestamps of all stop pulses that occur within a configurable range are grouped into one packet.

1.1 Features

- 4-channel common-start TDC with 8 ps resolution
- Standard Range: 218 µs (24-bit timestamp)
- Extended Range: 13.975 ms
- Bin size: 13 ps
- Double-pulse resolution for best results: 5 ns
- Double-pulse resolution without lost hits: 1.7 ns
- Dead time between groups: none
- Minimum interval between starts: 250 ns
- L0 FIFO: 15 words/channel
- L1 FIFO: 512 words/channel
- L2 FIFO: 8000 words
- PCIe 1.1 x1 with 200 MB/s throughput

1.2 Applications

The xTDC4 can be used in all time measurement applications where a common start setup with four channels is sufficient. For alternatives with more channels or more flexible grouping check our TDC website www.cronologic.de.

The xTDC4 is well suited for the following applications:

- Time-Of-Flight Mass Spectrometers (TOF-MS)
- LIDAR down to 2 mm resolution
- Reciprocal counters
- Coincidence measurements
- Quantum communication
- Time-Correlated Single Photon Counting (TCSPC)
- Synchronization of atomic clocks
- Fluorescence Lifetime Imaging Microscopy (FLIM)

2 Hardware

The xTDC4 is available as a PCIe plugin board (variant "-PCIe", see Ordering Information) or as a desktop solution (variant "-TBT"). The two variants are shown in Figure 2.1.

For a detailed overview of the hardware of the TBT-variant, please refer to our respective user guide.

Figure 2.1 Overview of the PCIe (left) and the TBT (right) variant of the xTDC4.

2.1 Installing the PCIe Board

The xTDC4 board can be installed in any PCIe-CEM slot with x1 or more lanes. Make sure that the PC is powered off and that the main power connector is disconnected while installing the board.

2.2 xTDC4 Inputs and Connectors

Figure 2.2 shows the location of the inputs on the slot bracket.

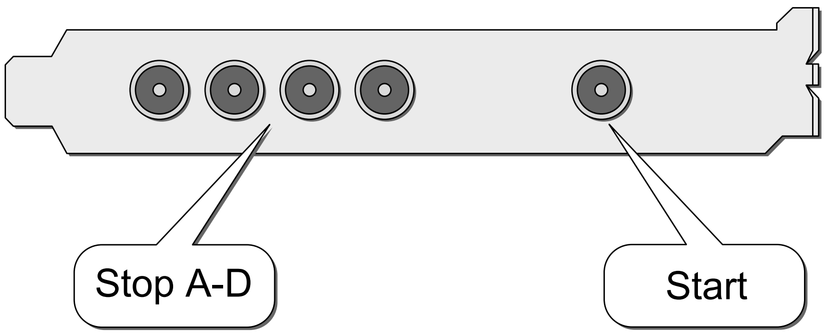

Figure 2.2 Input connectors of the xTDC4 on the PCIe bracket.

LEMO-00 connectors are used for input connection. The inputs are AC-coupled and have an impedance of 50 Ω. A schematic of the input circuit is shown in Figure 2.3. The digital threshold for any input can be

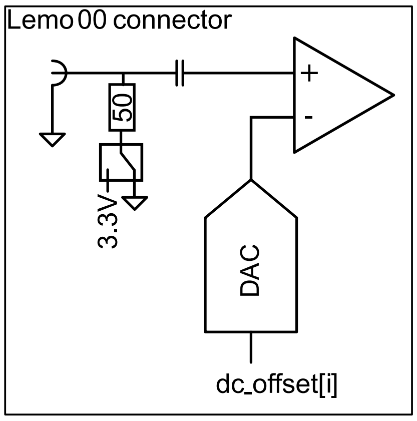

Figure 2.3 Input circuit for each of the input channels.

adjusted to comply with a multitude of single-ended signaling standards. The threshold can also be used to configure the input for either positive or negative pulses.

The connectors can also be used as outputs. DC-coupled output pulses for automatic internal triggering and control of external devices can be generated using the TiGer timing pattern generator. See Section 3.4 for details on the TiGer. Furthermore, for **Gen 1** boards three inter-board connectors can be found near the top edge of the xTDC4 board, as displayed in Figure 2.4. Connector J25 is reserved for future use. The pinout of connector J12 is shown in Table 2.1 and the pinout of connector J6 is depicted in Table 2.2. **Gen 2 boards do not possess these three connectors**.

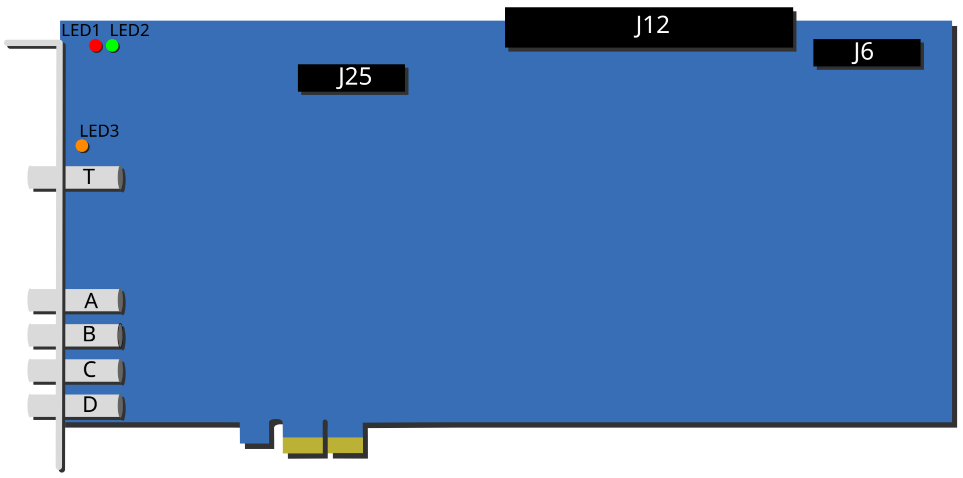

Figure 2.4 Schematic view of an xTDC4 board including the inter-board connectors. The inter-board connectors are only present on Gen 1 boards.

Pin	Name
1, 2	GND
3, 4	external CLK in N, external CLK in P
5, 6	GND
7, 8	reserved/NC
9, 10	GND
11, 12	reserved/NC
13, 14	GND
15, 16	reserved/NC
17, 18	GND
19, 20	reserved/NC
21, 22	GND
23, 24	reserved/NC
25, 26	GND
27, 28	reserved/NC
29, 30	GND
31, 32	reserved/NC
33, 34	GND

Table 2.1 Pinout of connector J12.

Pin	Name
1	+3.3 V
2 - 9	reserved/NC
10	GND

Table 2.2 Pinout of connector J6.

2.3 Status LEDs of the PCIe boards

Three status LEDs are present on the board, as seen in Figure 2.4.

- LED1 lights up red during the configuration of the FPGA and turns off afterward. If it stays permanently lit, the configuration failed.
- LED2 lights up green after the board is initialized by the driver and turns off when the device is closed by the software.
- LED3 lights up green when capture is started, yellow after the first start signal was detected and red when groups are missing.

3 xTDC4 Functionality

The xTDC4 is a “classic” common start time-to-digital converter.

It records the time difference between leading or trailing edges on the start input and the stop inputs. Each stop channel A-D can be enabled individually. The standard deviation of the timestamps is approx. 8 ps. The timestamps are recorded in integer multiples of a bin size of $5000/(3 \times 128) = 13.02083\overline{3}$ ps. The data acquisition can be limited to rising or falling signal transitions.

The maximum trigger rate on the start channel is 4 MHz.

3.1 Handling of Difficult Hits

Transitions of the input signals are called hits. To measure all hits with the maximum resolution the hits must fulfill all criteria set forth in this document. However, the xTDC4 does include mechanisms to provide as much information as possible for hits that fall out of these specs.

To reliably detect hits the signal has to be stable for at least 900 ps before and after the edge that is to be measured. Pulses as short as 250 ps are usually detected at full resolution but have a significant chance to be assigned to the wrong group or appear out of order. For these hits bit 7 in the data word is set. See Section 6.2 for more information on the data format.

Between multiple hits on a stop channel a dead time of approximately 5 ns is required for full resolution. Hits that are closer together will only be reported with a resolution of 5/6 ns = $833.\overline{3}$ ps. For these hits both bits 6 and 7 are set.

Data is copied from the 15-entry L0 FIFO to the larger downstream FIFOs at a rate of about 12 MHz per channel. If the L0 FIFO overflows the high resolution measurement of some hits will be discarded. In this case a measurement from an alternative TDC will be used that has a resolution of about 150 ps. For these hits bit 6 in the data word will be set.

3.2 Grouping and Events

In typical applications a start hit is followed by a multitude of stop hits. If grouping is enabled, the hits recorded are managed in groups (which are called “events” in some applications).

Figure 3.1 shows a corresponding timing diagram. The user can define the range of a group, i.e., the time window within which hits on the stop channels are recorded. Hits occurring outside that time window are discarded.

Different ranges can be set for each of the stop channels by setting corresponding values for `channel[i].start` and `channel[i].stop` values.

The values need to be set as multiples of the bin size and must not be negative.

$$0 \leq \mathit{start} \leq \mathit{stop} \leq 2^{16} - 1$$

If a second start is recorded within the range of a group, the current group is finished and a new group is started. Consecutive stops will be assigned to the new group (as long as they are within the group range).

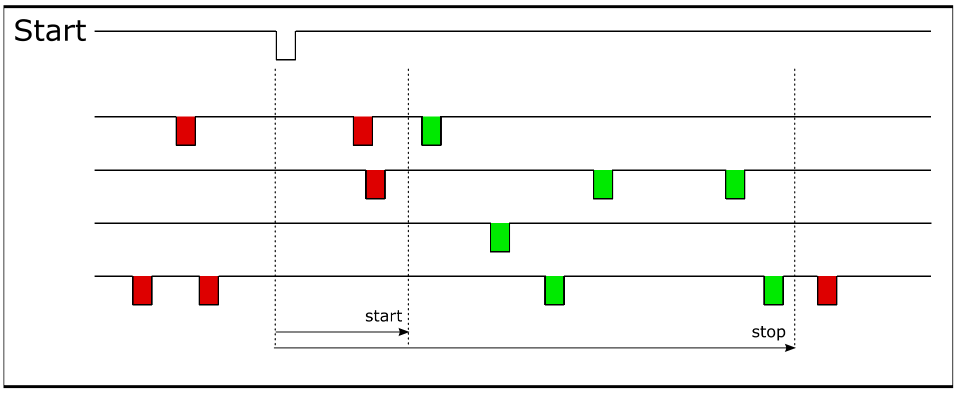

Figure 3.1 Acquired hits are merged to groups as explained in the text.

3.3 Auto-Triggering Function Generator

Some applications require internal periodic or random triggering. The xTDC4 function generator provides this functionality.

The delay between two trigger pulses of this trigger generator is the sum of two components: A fixed value M and a pseudo-random value with a range given by the exponent N.

The period is

$$T = M + [1...2^N] - 1$$

clock cycles with a duration of 4 ns per cycle. Here, $6 \leq M < 2^{32}$ and $0 \leq N < 32$.

The trigger can be used as a source for the TiGer unit (see Section 3.4) .

3.4 Timing Generator (TiGer)

Each digital LEMO-00 input can be used as an LVCMOS trigger output. The TiGer functionality can be configured independently for each connector. See Section 4.5.3 for a full description of the configuration options.

Figure 3.2 shows how the TiGer blocks are connected. They can be triggered by an OR of an arbitrary combination of inputs, including the auto-trigger . Each TiGer can drive its output to its corresponding LEMO connector. This turns the connector into an output.

The TiGer is DC coupled to the connector. Connected hardware must not drive any signals to connectors used as outputs, as doing so could damage both the xTDC4 and the external hardware. Pulses that are short enough for the input AC coupling are available as input signals to the xTDC4. This can be used to measure exact time differences between the generated output signals and input signals on other channels.

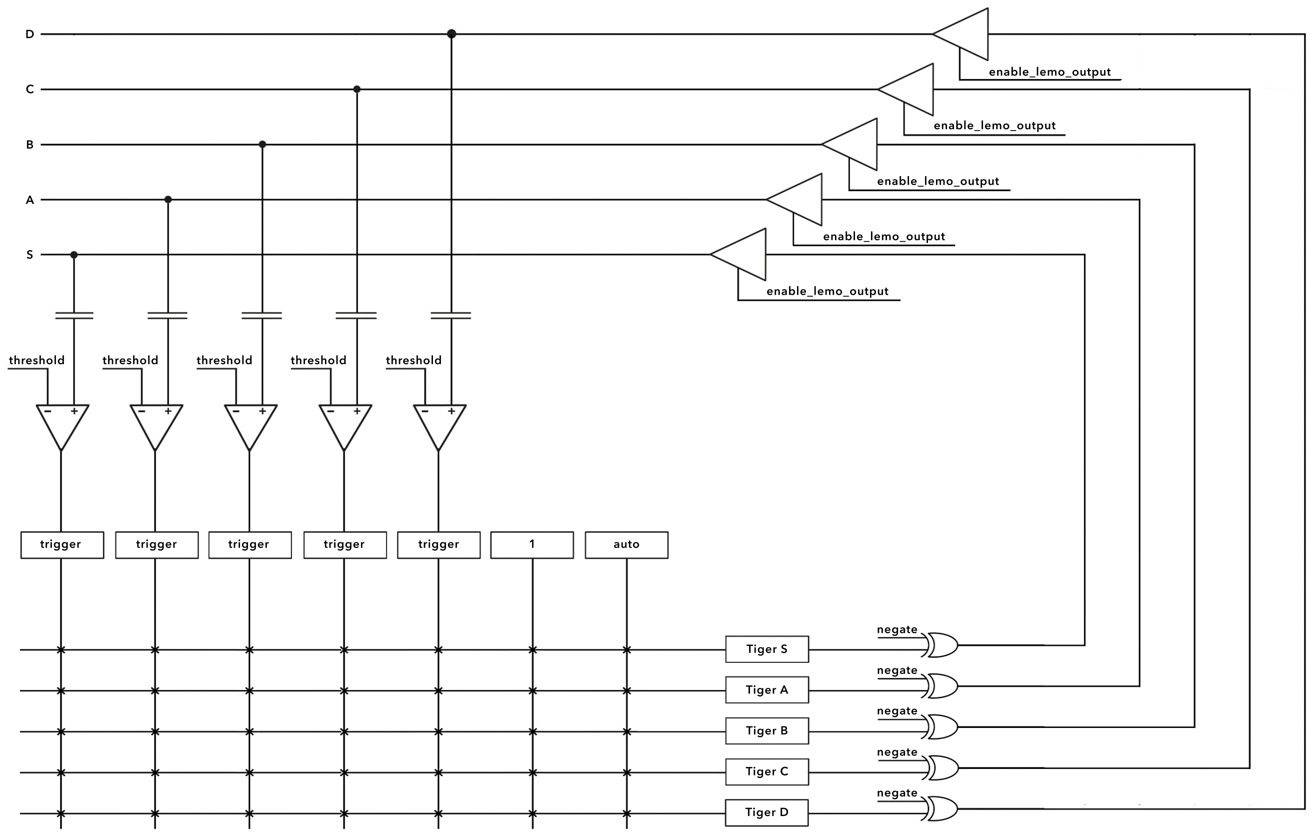

Figure 3.2 TiGer blocks can generate outputs that are also available on inputs.

4 Driver Programming API

The API is a DLL with C linkage.

The functions provided by the driver are declared in `xTDC4_interface.h` which must be included by your source code. You must tell your compiler to link with the file xTDC4_driver_64.lib. When running your program the dynamic link library containing the actual driver code must reside in the same directory as your executable or be in a directory included in the PATH variable. For Linux, it is provided only as a static library `libxtdc4_driver.a` The file for the DLL is called `xTDC4_driver_64.dll`.

All these files are provided with the driver installer that can be downloaded from the product website www.cronologic.de. By default, the installer will place the files into the directory `C:\Program Files\cronologic\xTDC4\driver`. A coding example can be found on github.com/cronologic-de/xtdc_babel.

4.1 Constants

`#define XTDC4_TDC_CHANNEL_COUNT 4`
The number of TDC input channels.

`#define XTDC4_TIGER_COUNT 5`
The number of timing generators. One for each TDC input and one for the start input.

`#define XTDC4_TRIGGER_COUNT 16`
The number of potential trigger sources for the timing generators. One for each TDC input, one for the Start input plus some specials. See Section 4.5.3 for details.

`#define XTDC4_OK 0`
Error codes are set by the API functions to this value if there has been no error. Other error codes can be found in `xTDC4_interface.h`

4.2 Driver Information

Even if there is no board present the driver revision can be queried using these functions.

`int xtdc4_get_driver_revision()`
Returns the driver version, same format as `xtdc4_static_info.driver_revision`. This function does not require an xTDC4 board to be present.

`const char* xtdc4_get_driver_revision_str()`
Returns the driver version including SVN build revision as a string.

`int xtdc4_count_devices(int *error_code, char **error_message)`
Returns the number of boards present in the system that are supported by this driver. Pointers to an error code and message variable have to be provided. If `error_code` does not equal `#define XTDC4_OK = 0`, the error message will contain what went wrong. E.g., the crono kernel was not properly installed.

4.3 Initialization

The card must be initialized first before reading data. Normally the process is to get the default initialization parameters and change some values. E.g., choose one of multiple cards by the index or use a larger buffer.

`int xtdc4_get_default_init_parameters(xtdc4_init_parameters *init)`

Sets up the standard parameters. Gets a set of default parameters for `xtdc4_init()`. This must always be used to initialize the `xtdc4_init_parameters` structure before modifying it and passing it to `xtdc4_init`.

`xtdc4_device xtdc4_init(xtdc4_init_parameters *params, int *error_code, char **error_message)`

Opens and initializes the xTDC4 board with the given index.

`error_code` must point to an integer where the driver can write the error code.

`error_message` must point to a pointer to char. The driver will allocate a buffer for zero-terminated error message and store the address of the buffer in the location provided by the user.

The parameter `params` is a pointer to a structure of type `xtdc4_init_parameters` that must be completely initialized by `get_default_init_parameters()`.

`int xtdc4_close(xtdc4_device *device)`

Closes the devices, releasing all resources.

4.3.1 Structure xtdc4_init_parameters

`int version`

The version number. Must be set to `XTDC4_API_VERSION`.

`int card_index`

The index in the list of xTDC4 boards that should be initialized.

There might be multiple boards in the system that are handled by this driver as reported by `xtdc4_count_devices`. This index selects one of them. Boards are enumerated depending on the PCIe slot. The lower the bus number and the lower the slot number the lower the card index.

`int board_id`

The global index in all cronologic devices.

This 8-bit number is filled into each packet created by the board and is useful if data streams of multiple boards will be merged. If only xTDC4 cards are used this number can be set to the `card_index`. If boards of different types that use a compatible data format are used in a system each board should get a unique ID. Can be changed with `int xtdc4_set_board_id (xtdc4_device *device, int board_id)`.

`uint64_t buffer_size[8]`

The minimum size of the DMA buffer.
If set to 0 the default size of 16 MByte is used. For the xTDC4, only the first entry is used.

`int buffer_type`

The type of buffer. Must be set to 0.

```
#define XTDC4_BUFFER_ALLOCATE 0
```

```
#define XTDC4_BUFFER_USE_PHYSICAL 1 // unsupported
```

`uint64_t buffer_address`
: This is set by `xtdc4_init()` to the start address of the reserved memory.

  The buffers will be allocated with the sizes given above. Make sure that the memory is large enough.

`int variant`
: Set to 0. Can be used to activate future device variants such as different base frequencies.

`int device_type`
: A constant for the different devices of cronologic `CRONO_DEVICE_*`.

  Initialized by `xtdc4_get_default_init_parameters()`. This value is identical to the PCI Device ID. Must be left unchanged.

```
#define CRONO_DEVICE_HPTDC        0x1
#define CRONO_DEVICE_NDIGO5G      0x2
#define CRONO_DEVICE_NDIGO250M    0x4
#define CRONO_DEVICE_xTDC4        0x6
#define CRONO_DEVICE_TIMETAGGER4  0x8
#define CRONO_DEVICE_XHPTDC8      0xC
#define CRONO_DEVICE_NDIGO6       0xD
```

`int dma_read_delay`
: The update delay of the write pointer after a packet has been sent over PCIe. Specified in multiples of 16 ns. Should not be changed by the user.

`int use_ext_clock`
: If set to 1, use external 10 MHz reference. If set to 0, use internal reference.

4.4 Status Information

4.4.1 Functions for Information Retrieval

The driver provides functions to retrieve detailed information on the board type, its configuration, settings, and state. The information is split according to its scope and the computational requirements to query the information from the board.

`int xtdc4_get_device_type(xtdc4_device *device)`
: Returns the type of the device as `CRONO_DEVICE_XTDC4`.

`const char* xtdc4_get_last_error_message(xtdc4_device *device)`
: Returns most recent error message.

`int xtdc4_get_fast_info(xtdc4_device *device, xtdc4_fast_info *info)`
: Returns fast dynamic information.

  This call gets a structure that contains dynamic information that can be obtained within a few microseconds.

`int xtdc4_get_param_info(xtdc4_device *device, xtdc4_param_info *info)`

Returns configuration changes.

Gets a structure that contains information that changes indirectly due to configuration changes.

`int xtdc4_get_static_info(xtdc4_device *device, xtdc4_static_info *info)`

Contains static information.

Gets a structure that contains information about the board that does not change during run time.

`int xtdc4_get_pcie_info(xtdc4_device *device, crono_pcie_info *pcie_info)`

Read PCIe information.

Gets a structure that contains information about the PCIe state, like correctable or uncorrectable errors.

`int xtdc4_clear_pcie_errors(xtdc4_device *device, int flags)`

Clear PCIe errors.

Only useful for PCIe debugging. `flags` is one of the following:

```
#define CRONO_PCIE_CORRECTABLE_FLAG     1
#define CRONO_PCIE_UNCORRECTABLE_FLAG   2
```

4.4.2 Structure xtdc4_static_info

This structure contains information about the board that does not change during run time. It is provided by the function `xtdc4_get_static_info()`.

`int size`

The number of bytes occupied by the structure.

`int version`

A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

`int board_id`

ID of the board.

`int driver_revision`

Encoded version number for the driver.

The lower three bytes contain a triple-level hierarchy of version numbers, e.g., `0x010103` encodes version 1.1.3.

The version adheres to the Semantic Versioning scheme as defined at https://semver.org. A change in the first digit generally requires a recompilation of user applications. Changes in the second digit denote significant improvements or changes that don't break compatibility and the third digit increments with minor bug fixes and similar updates that do not affect the API.

`int driver_build_revision`

Build number of the driver according to cronologic's internal versioning system.

`int firmware_revision`

Revision number of the FPGA configuration.

`int` **`board_revision`**

Board revision number.

The board revision number can be read from a register. It is a four-bit number that changes when the schematic of the board is changed. This should match the revision number printed on the board.

`int` **`board_configuration`**

Describes the schematic configuration of the board.

The same board schematic can be populated in multiple variants. This is an 8-bit code that can be read from a register.

`int` **`subversion_revision`**

Subversion revision ID of the FPGA configuration source code.

`int` **`chip_id`**

16 bit factory ID of the TDC chip.

`int` **`board_serial`**

Serial number.

Year and running number are concatenated in 8.24 format. The number is identical to the one printed on the silvery sticker on the board.

`unsigned int` **`flash_serial_high`**
`unsigned int` **`flash_serial_low`**

64-bit manufacturer serial number of the flash chip

`crono_bool_t` **`flash_valid`**

If not 0, the driver found valid calibration data in the flash on the board and is using it. This value is not applicable for the xTDC4.

`char` **`calibration_date[20]`**

DIN EN ISO 8601 string YYYY-MM-DD HH:MM of the time when the card was calibrated.

`char` **`bitstream_date[20]`**

DIN EN ISO 8601 string YYYY-MM-DD HH:MM of the time when the bitstream on the card was created.

4.4.3 Structure xtdc4_param_info

This structure contains configuration changes provided by `xtdc4_get_param_info()`.

`int` **`size`**

The number of bytes occupied by the structure.

`int` **`version`**

A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

`double` **`binsize`**

Bin size (in ps) of the measured TDC data.

`int` **`board_id`**

Board ID.

The board uses this ID to identify itself in the output data stream. The ID takes values between 0 and 255.

`int` **`channels`**

Number of TDC channels of the board.

Fixed at 4.

`int` **`channel_mask`**

Bit assignment of each enabled input channel.

Bit $0 \leq n < 4$ is set if channel n is enabled.

`int64_t` **`total_buffer`**

The total amount of DMA buffer in bytes.

`double` **`packet_binsize`**

For xTDC4 this is 1666.6 ps

`double` **`quantisation`**

Quantisation or measurement resolution. For the xTDC4 this is 13.0208 ps

4.4.4 Structure xtdc4_fast_info

`int` **`size`**

The number of bytes occupied by the structure.

`int` **`version`**

A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

`int` **`tdc_rpm`**

Speed of the TDC fan in rounds per minute. Reports 0 if no fan is present.

`int` **`fpga_rpm`**

Speed of the FPGA fan in rounds per minute. Reports 0 if no fan is present.

`int` **`alerts`**

Alert bits from the temperature sensor and the system monitor. Bit 0 is set if the TDC temperature exceeds 140 °C. In this case the TDC did shut down and the device needs to be reinitialized.

`int` **`pcie_pwr_mgmt`**

Always 0.

`int` **`pcie_link_width`**

Number of PCIe lanes the card uses. Should always be 10 for the xTDC4.

`int` **`pcie_max_payload`**

Maximum size in bytes for one PCIe transaction. Depends on system configuration.

4.4.5 Structure crono_pcie_info

`uint32_t` **`pwr_mgmt`**

Organizes power supply of PCIe lanes.

`uint32_t` **`link_width`**

Number of PCIe lanes that the card uses.

`uint32_t` **`max_payload`**

Maximum size in bytes for one PCIe transaction.

Depends on the system configuration.

`uint32_t` **`link_speed`**

Data rate of the PCIe card.

Depends on the system configuration.

`uint32_t` **`error_status_supported`**

Different from 0 if the PCIe error status is supported for this device.

`uint32_t` **`correctable_error_status`**

Correctable error status flags, directly from the PCIe config register.

Useful for debugging PCIe problems.

```
#define CRONO_PCIE_RX_ERROR                   1 << 0
#define CRONO_PCIE_BAD_TLP                    1 << 6
#define CRONO_PCIE_BAD_DLLP                   1 << 7
#define CRONO_PCIE_REPLAY_NUM_ROLLOVER        1 << 8
#define CRONO_PCIE_REPLAY_TIMER_TIMEOUT       1 << 12
#define CRONO_PCIE_ADVISORY_NON_FATAL         1 << 13
#define CRONO_PCIE_CORRECTED_INTERNAL_ERROR   1 << 14
#define CRONO_PCIE_HEADER_LOG_OVERFLOW        1 << 15
```

`uint32_t` **`correctable_error_status`**

Uncorrectable error status flags, directly from the PCIe config register.

Useful for debugging PCIe problems.

```
#define CRONO_PCIE_UNC_UNDEFINED                      1 << 0
#define CRONO_PCIE_UNC_DATA_LINK_PROTOCOL_ERROR       1 << 4
#define CRONO_PCIE_UNC_SURPRISE_DOWN_ERROR            1 << 5
#define CRONO_PCIE_UNC_POISONED_TLP                   1 << 12
#define CRONO_PCIE_UNC_FLOW_CONTROL_PROTOCOL_ERROR    1 << 13
#define CRONO_PCIE_UNC_COMPLETION_TIMEOUT             1 << 14
#define CRONO_PCIE_UNC_COMPLETER_ABORT                1 << 15
#define CRONO_PCIE_UNC_UNEXPECTED_COMPLETION          1 << 16
#define CRONO_PCIE_UNC_RECEIVER_OVERFLOW_ERROR        1 << 17
#define CRONO_PCIE_UNC_MALFORMED_TLP                  1 << 18
#define CRONO_PCIE_UNC_ECRC_ERROR                     1 << 19
#define CRONO_PCIE_UNC_UNSUPPROTED_REQUEST_ERROR      1 << 20
```

4.5 Configuration

The device is configured with a configuration structure. The user should first obtain a structure that contains the default settings of the device read from an on-board ROM, then modify the structure as needed for the user application and use the result to configure the device.

`int xtdc4_configure(xtdc4_device *device, xtdc4_configuration *config)`
Configures the `xtdc4_manager`.

`int xtdc4_get_current_configuration(xtdc4_device *device, xtdc4_configuration *config)`
Gets current configuration. Copies the current configuration to the specified config pointer.

`int xtdc4_get_default_configuration(xtdc4_device *device, xtdc4_configuration *config)`
Gets default configuration. Copies the default configuration to the specified config pointer.

4.5.1 Structure xtdc4_configuration

This is the structure containing the configuration information. It is used in conjunction with `xtdc4_get_default_configuration()`, `xtdc4_get_current_configuration()` and `xtdc4_configure()`.

It uses multiple substructures to configure various aspects of the board.

`int size`
The number of bytes occupied by the structure.

`int version`
A version number that is increased when the definition of the structure is changed. The increment can be larger than one to match driver version numbers or similar.

`int tdc_mode`
TDC mode. Can be grouped or continuous defined as follows:

```
#define XTDC4_TDC_MODE_GROUPED      0
#define XTDC4_TDC_MODE_CONTINUOUS   1
```

- Grouped functionality is explained in Section 3.2.
- Not applicable for the xTDC4.

`crono_bool_t start_rising`
Selects whether the rising or falling edge of the start signal is used to start a group.

`double dc_offset[XTDC4_TDC_CHANNEL_COUNT + 1]`
Set the threshold voltage for the input channels S, A ...D (see Figure 4.1).

- dc_offset[0] : threshold for channel Start
- dc_offset[1 - 4] : threshold for channels A ...D

The supported range is -1.27 to 1.13 V. This should be close to 50% of the height of the input pulse. Examples for various signaling standards are defined as follows.

Important Note: The supported range changed for driver release 1.10.7. That means, if you use a value for `dc_offset` outside the new supported range in your source code, the device configuration will adjust it automatically to the new supported range (e.g., a value of +1.18 V will be reduced to +1.13 V).

```
#define XTDC4_DC_OFFSET_P_NIM        +0.35
#define XTDC4_DC_OFFSET_P_CMOS       +1.13
#define XTDC4_DC_OFFSET_P_LVCMOS_33  +1.13
#define XTDC4_DC_OFFSET_P_LVCMOS_25  +1.13
#define XTDC4_DC_OFFSET_P_LVCMOS_18  +0.90
#define XTDC4_DC_OFFSET_P_TTL        +1.13
#define XTDC4_DC_OFFSET_P_LVTTL_33   +1.13
#define XTDC4_DC_OFFSET_P_LVTTL_25   +1.13
#define XTDC4_DC_OFFSET_P_SSTL_3     +1.13
#define XTDC4_DC_OFFSET_P_SSTL_2     +1.13
#define XTDC4_DC_OFFSET_N_NIM        -0.35
#define XTDC4_DC_OFFSET_N_CMOS       -1.27
#define XTDC4_DC_OFFSET_N_LVCMOS_33  -1.27
#define XTDC4_DC_OFFSET_N_LVCMOS_25  -1.25
#define XTDC4_DC_OFFSET_N_LVCMOS_18  -0.90
#define XTDC4_DC_OFFSET_N_TTL        -1.27
#define XTDC4_DC_OFFSET_N_LVTTL_33   -1.27
#define XTDC4_DC_OFFSET_N_LVTTL_25   -1.25
#define XTDC4_DC_OFFSET_N_SSTL_3     -1.27
#define XTDC4_DC_OFFSET_N_SSTL_2     -1.25
```

The inputs are AC coupled. Thus, the absolute voltage is not important for pulse inputs. It is the relative pulse amplitude that causes the input circuits to switch. The parameter must be set to the relative switching voltage for the input standard in use. If the pulses are negative, a negative switching threshold must be set and vice versa.

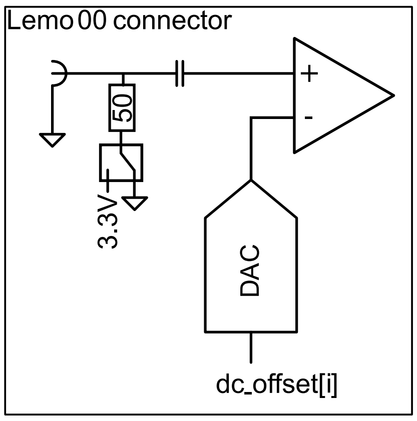

Figure 4.1 Input circuit for each of the input channels.

`xtdc4_trigger` **`trigger[XTDC4_TRIGGER_COUNT]`**

Configuration of the polarity of the external trigger sources (see Section 4.5.2). These are used as inputs for the TiGer blocks and as inputs to the time measurement unit.

`xtdc4_tiger_block` **`tiger_block[XTDC4_TIGER_COUNT]`**

Configuration of the timing generators (TiGer, see Section 4.5.3).

Index 0 refers to the Start channel; indices 1 through 4 to the Stop channels A through D.

`xtdc4_channel` **`channel[XTDC4_TDC_CHANNEL_COUNT]`**

Configuration of the TDC channels.

`xtdc4_lowres_channel`
`xtdc4_lowres_channel[XTDC4_LOWRES_CHANNEL_COUNT]`

Only applicable to the xTDC4-S. Configures the additional digital low-res inputs.

`uint32_t` **`auto_trigger_period`**
`uint32_t` **`auto_trigger_random_exponent`**

Create a trigger either periodically or randomly. There are two parameters

$$M = \texttt{auto\_trigger\_period}$$
$$N = \texttt{random\_exponent}$$

that result in a distance between triggers of T clock cycles.

$$T = M + [1...2^N] - 1$$
$$6 \leq M < 2^{32}$$
$$0 \leq N < 32$$

There is no enable or reset. The auto-trigger is running continuously. The usage of this trigger can be configured in the TiGer block source field.

4.5.2 Structure xtdc4_trigger

For each input, this structure determines whether rising or falling edges on the inputs create trigger events for the TiGer blocks.

`crono_bool_t` **`falling`**
`crono_bool_t` **`rising`**

Select for which edges a trigger event is created inside the FPGA. The xTDC4can output measurements with a reduced bin size of 5/6 ns = 833.333 ps for one or both edges of input signals. See section 3.1 for more information on hits with varying resolution. Use xTDC4_channel.rising on page 25 to select which edge is measured with full resolution. The edge that is selected for full resolution measurement must also be enabled for low resolution measurement.

4.5.3 Structure xtdc4_tiger_block

See Section 3.4 for additional information.

`crono_bool_t` **`enable`**

Activates the timing generator (TiGer).

`crono_bool_t` **`negate`**

Inverts output polarity. Default is set to `false`.

`crono_bool_t` **`retrigger`**

Enables re-trigger setting.

If enabled the timer is reset to the value of the *start* parameter, whenever the input signal is set while waiting to reach the *stop* time.

`crono_bool_t` **`extend`**

Not implemented.

`crono_bool_t` **`enable_lemo_output`**

Enables the LEMO output. Drive the TiGer signal to the corresponding LEMO connector as an output. This is DC coupled, so make sure that you do not connect any devices as inputs. Pulses created by the TiGer are visible at the inputs of the xTDC4 and can be measured again to get the exact timing.

`uint32_t` **`start`**
`uint32_t` **`stop`**

In multiples of 20/3 ns = 6.666 ns The time during which the TiGer output is set, relative to the trigger input.

The parameters `start` and `stop` must fulfill the following conditions

$$0 \leq \mathtt{start} \leq \mathtt{stop} \leq 2^{16} - 1 \,.$$

If retriggering is enabled, the timer is reset to the value of the start parameter whenever the input signal is set while waiting for the stop time.

`int` **`sources`**

A bit mask with a bit set for all trigger sources that can trigger this TiGer block. Default is `XTDC4_TRIGGER_SOURCE_S`.

```
#define XTDC4_TRIGGER_SOURCE_NONE   0x00000000
#define XTDC4_TRIGGER_SOURCE_S      0x00000001
#define XTDC4_TRIGGER_SOURCE_A      0x00000002
#define XTDC4_TRIGGER_SOURCE_B      0x00000004
#define XTDC4_TRIGGER_SOURCE_C      0x00000008
#define XTDC4_TRIGGER_SOURCE_D      0x00000010
#define XTDC4_TRIGGER_SOURCE_AUTO   0x00004000
#define XTDC4_TRIGGER_SOURCE_ONE    0x00008000
```

4.5.4 Structure xtdc4_channel

Contains TDC channel settings.

`crono_bool_t` **`enabled`**

Enable the TDC channel.

`crono_bool_t` **`rising`**

Select which edge of the signal is used for full resolution measurements. `xtdc4_trigger.rising` and `xtdc4_trigger.falling` described on Page 24 are used to select which edges are recorded for low resolution measurement. The edge that is selected for full resolution measurement must also be enabled for low resolution measurement. See Section 3.1 for more information on hits with varying resolution.

`crono_bool_t` **`cc_enable`**

Enable carry chain TDC. This is set to *true* by default and should be left unchanged.

`crono_bool_t` **`cc_same_edge`**

Sets whether the carry chain TDC records the same or opposite edge as the TDC chip. If the same edge is selected, that carry chain TDC acts as a backup if the chip misses hits due to FIFO overflows or short input pulses. If opposite edges are selected, both edges of a pulse can be measured with reasonable resolution. See Section 3.1 for more information.

`crono_bool_t` **`ths788_disable`**

Disable full resolution timestamps. This is set to `false` by default and should be left unchanged.

`uint32_t` **`start`**
`uint32_t` **`stop`**

Veto function for grouping of hits into packets in multiples of the binsize. Only hits between start and stop are read out. The parameters must adhere to the following relations:

$$0 \leq \texttt{start} \leq \texttt{stop} < 2^{30}$$

5 Run Time Control

5.1 Run Time Control

Once the devices are configured the following functions can be used to control the behavior of the devices. All of these functions return quickly with very little overhead, but they are not guaranteed to be thread safe.

`int` **`xtdc4_start_capture(`**`xtdc4_device` **`*device)`**

Start data acquisition.

`int` **`xtdc4_pause_capture(`**`xtdc4_device` **`*device)`**

Pause a started data acquisition.

`pause` and `continue` have less overhead than start and stop but don't allow for a configuration change.

`int` **`xtdc4_continue_capture(`**`xtdc4_device` **`*device)`**

Call this to resume data acquisition after a call to `xtdc4_pause_capture()`.

`pause` and `continue` have less overhead than start and stop but don't allow for a configuration change.

`int` **`xtdc4_stop_capture(`**`xtdc4_device` **`*device)`**

Stop data acquisition.

`int` **`xtdc4_start_tiger(`**`xtdc4_device` **`*device)`**
`int` **`xtdc4_stop_tiger(`**`xtdc4_device` **`*device)`**

Start and stop the timing generator. This can be done independently of the state of the data acquisition.

5.2 Readout

The device provides a stream of packets, that are read in batches. A batch of packets is provided to the application, it processes them, by storing important information in other structures. The batch that were processed need to be acknowledged, so that the device can reuse the memory of these for the next data. That means processing should be fast.

```
timetagger4_read_in in;
// automatically acknowledge all data as processed
in.acknowledge_last_read = 1;
volatile crono_packet* p = read_data.first_packet;
timetagger4_read_out out;
int status = timetagger4_read(device, &in, &out);
if (status == CRONO_READ_OK) {
    while (p <= read_data.last_packet) {
        processPacket(p);
        p = crono_next_packet(p);
    }
}
```

`int` **`xtdc4_acknowledge(`**`xtdc4_device` **`*device,`** `crono_packet` **`*packet)`**

Acknowledges the processing of the last read block. This is only necessary if `xtdc4_read()` is not called with `in.acknowledge_last_read` set.

This feature allows to either free up partial DMA space early if there will be no call to `xtdc4_read()` anytime soon. It also allows keeping data over multiple calls to `xtdc4_read()` to avoid unnecessary copying of data.

`int` **`xtdc4_read(`**`xtdc4_device` **`*device,`** `xtdc4_read_in` **`*in,`**
`xtdc4_read_out` **`*out)`**

Return a pointer to an array of captured data in read_out. The result contains a batch of packets of type xtdc4_packet. The batch is described by `first_packet` and `last_packet` in the `xtdc4_read_in` structure.

`read_in` provides parameters to the driver. A call to this method automatically allows the driver to reuse the memory returned by the previous call if `read_in.acknowledge_last_read` is set.

Returns an error code as defined in the structure `xtdc4_read_out`.

`crono_packet` **`crono_next_packet(`**`crono_packet` **`*packet)`**

Iterates to the next packet in the batch.

5.2.1 Input Structure xtdc4_read_in

`crono_bool_t` **`acknowledge_last_read`**

If set `xtdc4_read()` automatically acknowledges packets from the last read. Otherwise, `xtdc4_acknowledge()` needs to be called explicitly by the user.

5.2.2 Input Structure xtdc4_read_out

`crono_packet` **`*first_packet`**

Pointer to the first packet that was captured by the call of `xtdc4_read()`.

`crono_packet` **`*last_packet`**

Address of header of the last packet in the buffer. This packet is still valid, all data after this packet is invalid.

`int` **`error_code`**

Assignments of the error codes.

```
#define CRONO_READ_OK              0
#define CRONO_READ_NO_DATA         1
#define CRONO_READ_INTERNAL_ERROR  2
#define CRONO_READ_TIMEOUT         3
```

`const char` **`*error_message`**

The last error in human-readable form, possibly with additional information about the error.

6 Output Data Format

6.1 Memory Management

The *host buffer* is memory on the host's system in which the data recorded by the xTDC4 is stored until it is acknowledged by the user.

The host buffer is managed by the DMA (direct memory access) driver. The DMA driver can only ever write to the host buffer if enough memory is free. That means, new packets will never overwrite old packets, unless they have been acknowledged.

If the host buffer is full, data may be lost. Then, the `CRONO_PACKET_FLAG_HOST_BUFFER_FULL` bit of `crono_packet::flags` is set. To ensure that this does not happen, the user must acknowledge packets fast enough by the analysis software. Note that data only has been lost due to a full host buffer if the `CRONO_PACKET_FLAG_TRIGGER_MISSED` bit of `crono_packet::flags` is set.

6.1.1 Acknowledge Packets

A packet in the host buffer will only be overwritten if it has been acknowledged. This can be done manually by the user by calling `ndigo_acknowledge()` or automatically by the driver if in the call of `ndigo_read()`, `acknowledge_last_read` of the `ndigo_read_in` structure `in` was set to `true` (see Section 5).

Acknowledging a packet acknowledges all previous packets as well.

Be aware that acknowledging a packet *immediately* invalidates it, and it immediately becomes unsafe to attempt accessing its content.

6.1.2 xTDC4-Internal Memory Layout

The xTDC4 uses internal FIFO (first-in, first-out) memories. In one of these FIFOs, referred to as the DMA FIFO, packets that are ready to be sent to the host system are buffered. If the DMA FIFO is full at any point, the affected packets `CRONO_PACKET_FLAG_DMA_FIFO_FULL` bit of `crono_packet::flags` is set. This does not mean that data has been necessarily lost. Only if the `CRONO_PACKET_FLAG_TRIGGER_MISSED` bit is set has data been lost.

6.2 Output Structure crono_packet

Output of a read call list is a group of `crono_packet` structures. Which have a variable length. The structure contains the following fields.

`uint8_t` **`channel`**
    Index of the source channel of the data. Pseudo channel 15 is used for rollovers.

`uint8_t` **`card`**
    Identifies the source card in case there are multiple boards present. Defaults to 0 if no value is assigned to the parameter `board_id` in structure `timetagger4_init_parameters`.

`uint8_t` **`type`**

The data stream consists of 32-bit unsigned data as signified by `CRONO_PACKET_TYPE_32_BIT_UNSIGNED = 6`.

`uint8_t` **`flags`**

Bit field of TIMETAGGER4 _PACKET_FLAG_* bits:

`#define XTDC4_PACKET_FLAG_ODD_HITS 1`
If this bit is set, the last data word in the data array consists of one timestamp only which is located in the lower 32 bits of the 64-bit data word (little endian).

`#define XTDC4_PACKET_FLAG_SLOW_SYNC 2`
Timestamp of a hit is above the range of 8-bit rollover number and 24-bit hit timestamp. The group is closed, all other hits are ignored.

`#define XTDC4_PACKET_FLAG_START_MISSED 4`
The trigger unit has discarded packets due to a full FIFO because the data rate is too high. Starts are missed and stops are potentially in wrong groups.

`#define XTDC4_PACKET_FLAG_SHORTENED 8`
The trigger unit has shortened the current packet due to a full pipeline FIFO because the data rate is too high. Stops are missing in the current packet.

`#define XTDC4_PACKET_FLAG_DMA_FIFO_FULL 16`
The internal DMA FIFO was full. This is caused either because the data rate is too high on too many channels. Packet loss is possible.

`#define XTDC4_PACKET_FLAG_HOST_BUFFER_FULL 32`
The host buffer was full. Might result in dropped packets. This is caused either because the data rate is too high or by data not being retrieved fast enough from the buffer. Solutions are increasing buffer size if the overload is temporary or by avoiding or optimizing any additional processing in the code that reads the data.

`uint32_t` **`length`**

Number of 64-bit elements (each containing up to 2 TDC hits) in the data array. The number of hits contained is equal to `2 * length - (flags & PACKET_FLAG_ODD_HITS) ? 1 : 0`.

`uint64_t` **`timestamp`**

Coarse timestamp of the start pulse. Values are given in multiples of $5/3 = 1.\overline{6}$ ns.

`uint64_t` **`data[1]`**

Contains the TDC hits as a variable length array (length can be zero). The user can cast the array to `uint32_t*` to directly operate on the TDC hits. For the number of hits, see length. Structure of one hit (32 bit):

bits	31 to 8	7 to 4	3 to 0
content	TDC DATA	FLAGS	CHN

The timestamp of the hit is stored in bits 31 down to 8 in multiples of $5/(3 * 128) = 13.02083\overline{3}$ ps

```
uint32_t timestamp  = (hit >> 8) & 0xF;
uint32_t flags      = (hit >> 4) & 0xF;
uint32_t channel    =  hit       & 0xF;
```

Bits 7 down to 4 are hit flags and have the following definitions:

- `#define XTDC4_HIT_FLAG_FPGA_MISSING` 8 ↔ Bit 7
  `#define XTDC4_HIT_FLAG_COARSE_TIMESTAMP` 4 ↔ Bit 6
  Bit 7, 6: Resolution of this measurement (see Section 3.1).

bit 7	bit 6	Measurement Type
0	0	Normal full resolution measurement.
0	1	Measurement performed with carry chain TDC at about 150 ps resolution.
1	0	Full resolution measurement that might in the wrong place in the data stream.
1	1	Measurement with only 5/6 ns = $833.\overline{3}$ ps resolution.

- `#define XTDC4_HIT_FLAG_TIME_OVERFLOW` 2 ↔ Bit 5
  Bit 5: Rollover. The time since start pulse exceeded the 24-bit range that can be encoded in a data word. This word does not encode a measurement. Instead, the readout software should increment a rollover counter that can be used as the upper bits of consecutive time stamps. The counters should be reset for each packet. The total offset of a hit in picoseconds can be computed by

$$\begin{aligned}\Delta T_{hit} = &(\#\text{rollovers} \times \texttt{xtdc4\_static\_info.rollover\_period} + \text{TDC\_DATA}_{\text{hit}}) \\ &\times \texttt{xtdc4\_param\_info.binsize}\end{aligned}$$

- `#define XTDC4_HIT_FLAG_RISING` 1 ↔ Bit 4
  Bit 4: Set if this hit is a rising edge. Otherwise, this word belongs to a falling edge. The channel number is given in the lowest nibble of the data word.
  A value of 0 corresponds to channel A, a value of 3 to channel D.

7 Code Example

The following C++ source code shows how to initialize an xTDC4 board, configure it and loop over incoming packets.

If you are reading this documentation in portable document format (PDF), the source code of the C example is also embedded as an attachment to the file. You can open it in an external viewer or save it to disk by clicking on it.

```cpp
// xtdc4_user_guide_example.cpp : Example application for the xTDC4
#include "xTDC4_interface.h"
#include "stdio.h"
#include <windows.h>

typedef unsigned int uint32;
typedef unsigned __int64 uint64;

xtdc4_device * initialize_xtdc4(int buffer_size, int board_id, int card_index) {
        // prepare initialization
        xtdc4_init_parameters params;

        xtdc4_get_default_init_parameters(&params);
        params.buffer_size[0] = buffer_size;        // size of the packet buffer
        params.board_id = board_id;                 // value copied to "card" field of every packet, allowed range 0..255
        params.card_index = card_index;             // which of the xTDC4 board found in the system to be used

        int error_code;
        const char * err_message;
        xtdc4_device * device = xtdc4_init(&params, &error_code, &err_message);
        if (error_code != CRONO_OK) {
                printf("Could not init xTDC4 compatible board: %s\n", err_message);
                return nullptr;
        }
        return device;
}

int configure_xtdc4(xtdc4_device * device) {
        // prepare configuration
        xtdc4_configuration config;

        // fill configuration data structure with default values
        // so that the configuration is valid and only parameters
        // of interest have to be set explicitly
        xtdc4_get_default_configuration(device, &config);

        // set config of the 4 TDC channels
        for (int i = 0; i < XTDC4_TDC_CHANNEL_COUNT; i++)
        {
                // enable recording hits on TDC channel
                config.channel[i].enabled = true;

                // define range of the group
                config.channel[i].start = 0;    // range begins right after ↩
                   start pulse
                config.channel[i].stop = 30000; // recording window stops ↩
                   after ~390 ns (30000 * 13.02ps)

                // measure only falling edge
                config.trigger[i + 1].falling = 1;
                config.trigger[i + 1].rising = 0;
        }

        // start group on falling edges on the start channel 0
        config.trigger[0].falling = 1;  // enable packet generation on ↩
           falling edge of start pulse
        config.trigger[0].rising = 0;   // disable packet generation on ↩
           rising edge of start pulse

        // generate an internal 200 kHz trigger
        config.auto_trigger_period = 750;           // multiples of 6.666 ns
        config.auto_trigger_random_exponent = 0;

        // setup TiGeR
        // sending a signal to the LEMO outputs (and to the TDC on the same ↩
           channel)
        // requires proper 50 Ohm termination on the LEMO output to work ↩
           reliably

        // use 200 kHz auto trigger to generate
        // a 200 kHz signal with 12 ns pulse width on LEMO output Start
        config.tiger_block[0].enable = 1;
        config.tiger_block[0].start = 0;
        config.tiger_block[0].stop = config.tiger_block[0].start + 1;
        config.tiger_block[0].negate = 0;
        config.tiger_block[0].retrigger = 0;
        config.tiger_block[0].extend = 0;
        config.tiger_block[0].enable_lemo_output = 1;
        config.tiger_block[0].sources = XTDC4_TRIGGER_SOURCE_AUTO;

        // if TiGeR is used for triggering with positive pulses
        config.dc_offset[0] = XTDC4_DC_OFFSET_P_LVCMOS_18;

        // write configuration to board
        return xtdc4_configure(device, &config);
}

double get_binsize(xtdc4_device * device) {
        xtdc4_param_info parinfo;
        xtdc4_get_param_info(device, &parinfo);
        return parinfo.binsize;
}
```

```c

void print_device_information(xtdc4_device * device) {
        // print board information
        xtdc4_static_info staticinfo;
        xtdc4_get_static_info(device, &staticinfo);
        printf("Board Serial        : %d.%d\n", staticinfo.board_serial >> 24, staticinfo.board_serial & 0xffffff);
        printf("Board Configuration : 0x%x\n", staticinfo.board_configuration);
        printf("Board Revision      : %d\n", staticinfo.board_revision);
        printf("Firmware Revision   : %d.%d\n", staticinfo.firmware_revision, staticinfo.subversion_revision);
        printf("Driver Revision     : %d.%d.%d\n", ((staticinfo.driver_revision >> 16) & 255), ((staticinfo.driver_revision >> 8) & 255), (staticinfo.driver_revision & 255));
        printf("Driver SVN Revision : %d\n", staticinfo.driver_build_revision);
        printf("\nTDC binsize         : %0.2f ps\n", get_binsize(device));
}

void print_hit(uint32 hit, double binsize) {
        // extract channel number (A-D)
        char channel = 65 + (hit & 0xf);

        // extract hit flags
        int flags = (hit >> 4 & 0xf);

        // extract hit timestamp
        int ts_offset = (hit >> 8 & 0xffffff);

        // TDC bin size is 13.02 ps. Convert timestamp to ns.
        double ts_offset_ns = ts_offset;
        ts_offset_ns *= binsize / 1000.0;

        printf("Hit @Channel %c - Flags %d - Offset %u (raw) / %.1f ns\n", channel, flags, ts_offset, ts_offset_ns);
}

_int64 process_packet(_int64 group_abs_time_old, volatile crono_packet *p, int update_count, double binsize) {
        // do something with the data, e.g. calculate current rate
        _int64 group_abs_time = p->timestamp;
        // group timestamp increments at 600 MHz
        double rate = (600000000 / ((double)(group_abs_time - group_abs_time_old) / (double)update_count));
        printf("\r%.2f kHz ", rate / 1000.0);

        // ...or print hits (not a good idea at high data rates,
        printf("Card %d - Flags %d - Length %d - Type %d - TS %llu\n", p->card, p->flags, p->length, p->type, p->timestamp);

        // There fit two hits into every 64 bit word.
        // The flag with weight 1 tells us, whether the number of hits in the packet is odd
        int hit_count = 2 * (p->length);
        if ((p->flags & 0x1) == 1)
                hit_count -= 1;

        uint32* packet_data = (uint32*)(p->data);
        for (int i = 0; i < hit_count; i++)
        {
                print_hit(packet_data[i], binsize);
        }
        printf("\n\n");
        return group_abs_time;
}

int main(int argc, char* argv[]) {
        printf("cronologic xtdc4_user_guide_example using driver: %s\n", ↩
           xtdc4_get_driver_revision_str());

        xtdc4_device * device = initialize_xtdc4(8 * 1024 * 1024, 0, 0);

        int status = configure_xtdc4(device);
        if (status != CRONO_OK) {
                printf("Could not configure xTDC4: %s", ↩
                   xtdc4_get_last_error_message(device));
                xtdc4_close(device);
                return status;
        }

        print_device_information(device);

        // configure readout behaviour
        xtdc4_read_in read_config;
        // automatically acknowledge all data as processed
        // on the next call to xtdc4_read()
        // old packet pointers are invalid after calling xtdc4_read()
        read_config.acknowledge_last_read = 1;

        // structure with packet pointers for read data
        xtdc4_read_out read_data;

        // start data capture
        status = xtdc4_start_capture(device);
        if (status != CRONO_OK) {
                printf("Could not start capturing %s", ↩
                   xtdc4_get_last_error_message(device));
                xtdc4_close(device);
                return status;
        }

        // start timing generator
        xtdc4_start_tiger(device);

        // some book keeping
        int packet_count = 0;
        int empty_packets = 0;
        int packets_with_errors = 0;
        bool last_read_no_data = false;
        _int64 group_abs_time = 0;
        _int64 group_abs_time_old = 0;
        int update_count = 100;
        double binsize = get_binsize(device);

        printf("Reading packets:\n");
        // read 10000 packets
        while (packet_count < 10000)
        {
                // get pointers to acquired packets
                status = xtdc4_read(device, &read_config, &read_data);
                if (status != CRONO_OK) {
                        Sleep(100);
                        printf(" - No data! -\n");
                }
                else
                {
                        // iterate over all packets received with the last ↩
                           read
                        volatile crono_packet* p = read_data.first_packet;
                        while (p <= read_data.last_packet)
                        {
                                // printf is slow, so this demo only ↩
                                   processes every 1000th packet
                                // your application would of course collect ↩
                                   every packet
                                if (packet_count % update_count == 0) {
                                        group_abs_time = process_packet(↩
                                           group_abs_time, p, update_count, ↩
                                           binsize);
                                }
                                p = crono_next_packet(p);
                                packet_count++;
                        }
                }
        }

        // shut down packet generation and DMA transfers
        xtdc4_stop_capture(device);

        // deactivate xTDC4
        xtdc4_close(device);

        return 0;
}
```

8 Technical Data

Each board is tested against the values listed in the columns “Min” and “Max”. “Typical” is the mean value of the first 10 boards produced or a value that is set by design.

8.1 TDC Characteristics

8.1.1 TDC measurement Characteristics

Symbol	Parameter	Min	Typical	Max	Units
INL	Integral nonlinearity			1	bins
DNL	Differential nonlinearity			0.5	bins
t_{Bin}	Binsize		5000/384 $13.0208\overline{3}$		ps ps
t_{DPfull}	Interval between edges for full resolution	5			ns
t_{DPCC}	Interval between edges for carry chain TDC	10			ns
t_{DPlow}	Interval between edges for lowres measurements	1.8			ns
Δt_{Start}	Interval between consecutive starts	250			ns
t_{Range}	Measurement range using hits only			$2^{24}-1$ 218	bins µs
$t_{Extended}$	Extended range using rollovers			$2^{30}-1$ 14	ms ms
$f_{Readout}$	Readout rate			48	MHits/s

8.1.2 Oscillator Time Base

Symbol	Parameter	Min	Typical	Max	Units
ΔT	Stability in temperature range −20 °C to 70 °C[1]			10	ppb
F	Initial calibration		<300	500	ppb
$\Delta F/F_1$	Aging first year			100	ppb
$\Delta F/F_{20}$	All inclusive aging 20 years			1000	ppb
	Warm-up[2]			3	min.

[1] Over –40 °C to +85 °C; relative to stabilized frequency after 1 hour of continuous operation

[2] @+25 °C; within ±100 ppb of F, where F is the stabilized frequency reached after 1 hour of continuous operation

8.2 Electrical Characteristics

8.2.1 Power Supply

Symbol	Parameter	Min	Typical	Max	Units
P_{total}	Total power consumption			10	W
$VCC_{3.3}$	PCIe 3.3 V rail power supply voltage	3.1	3.3	3.5	V
$I_{3.3}$	PCIe 3.3 V rail input current			600	mA
VCC_{12}	PCIe 12 V rail power supply voltage	11.1	12.0	12.9	V
I_{12}	PCIe 12 V rail input current			650	mA
VCC_{aux}	PCIe 3.3 V_{Aux} rail power supply voltage		3.3		V
I_{aux}	PCIe 3.3 V_{Aux} rail input current		0		mA

8.2.2 TDC Inputs

The xTDC4's inputs are single-ended AC-coupled with 50 Ω termination.

Symbol	Parameter	Min	Typical	Max	Units
V_{Base}	Input Baseline	0		5	V
$V_{Threshold}$	Trigger Level	$V_{Base} - 1.27$		$V_{Base} + 1.13$	V
t_{Pulse}	Pulse Length	2	5	200	ns
t_{Rise}	Pulse Edge 20% to 80%			10	ns
t_{Fall}	Pulse Edge 80% to 20%			10	ns
Z_P	Input Impedance		50		Ω
I_{Term}	Termination Current	–50	–20	50	mA

All inputs are AC-coupled. The inputs have very high input bandwidth requirements and therefore there are no circuits that provide over-voltage protection for these signals. Any voltage on the inputs above 5 V or below −5 V relative to the voltage of the slot cover can result in permanent damage to the board.

Keep in mind, that the input baseline V_{Base} is affected by the ratio of pulse length t_{Pulse} to average pulse distance (for continuous signals the term is called duty cycle).

Make sure not to drive the inputs when the connector is configured as a TiGer output.

See Section 3.4.

8.3 Information Required by DIN EN 61010-1

8.3.1 Manufacturer

The xTDC4 is a product of:

cronologic GmbH & Co. KG
Jahnstraße 49
60318 Frankfurt

Germany HRA 42869 beim Amtsgericht Frankfurt/M

VAT-ID: DE235184378
PCI Vendor ID: 0x1A13

8.3.2 Intended Use and System Integration

The devices are not ready to use as delivered by cronologic. It requires the development of specialized software to fulfill the application of the end-user. The device is provided to system integrators to be built into measurement systems that are distributed to end users. These systems usually consist of the xTDC4, a main board, a case, application software and possibly additional electronics to attach the system to some type of detector. They might also be integrated with the detector.

The xTDC4 is designed to comply with DIN EN 61326-1 when operated on a PCIe compliant main board housed in a properly shielded enclosure. When operated in a closed standard compliant enclosure the device does not pose any hazards as defined by EN 61010-1.

Radiated emissions, noise immunity, and safety highly depend on the quality of the enclosure. It is the responsibility of the system integrator to ensure that the assembled system is compliant to applicable standards of the country that the system is operated in, especially regarding user safety and electromagnetic interference.

When handling the board, adequate measures must be taken to protect the circuits against electrostatic discharge (ESD). All power supplied to the system must be turned off before installing the board.

8.3.3 Environmental Conditions for Storage

The board shall be stored between operation under the following conditions:

Symbol	Parameter	Min	Typical	Max	Units
T_{store}	ambient temperature	–30		60	°C
RH_{store}	relative humidity at 31°C noncondensing	10		70	%

8.3.4 Environmental Conditions for Operation

The board is designed to be operated under the following conditions:

Symbol	Parameter	Min	Typical	Max	Units
T_{oper}	ambient temperature	5		40	°C
RH_{oper}	relative humidity at 31°C	20		75	%

WARNING: Do not connect any DC-coupled inputs to a channel while the TiGer of that channel is configured as an output (see Section 3.4). Doing so could permanently damage the xTDC4 and the external hardware.

8.3.5 Cooling

The xTDC4 in its base configuration has passive cooling that requires a certain amount of air-flow. If the case design can't provide enough air-flow to the board, a slot cooler like Zalman ZM-SC100 can be placed next to the board. Active cooling is also available as an option for the board.

8.3.6 Recycling

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

The xTDC4 belongs to category 6, "Kleine Geräte der Informations- und Telekommunikationstechnik für die ausschließliche Nutzung in anderen als privaten Haushalten." Devices sold before 2018 belong to category 9, "Überwachungs und Kontrollinstrumente für ausschließlich gewerbliche Nutzung." The last owner of a xTDC4 must recycle it, treat the board in compliance with §11 and §12 of the German ElektroG, or return it to the manufacturer's address listed on Page 40.

9 Ordering Information

`xTDC4-PCIe`
: PCIe CEM x1 plugin board.

`xTDC4-TBT`
: External device connection to Thunderbolt ports.

10 Revision History

User Guide 1.10.4 as of 2024-09-19
cronologic GmbH & Co. KG
Jahnstraße 49
60318 Frankfurt am Main
Germany

10.1 Firmware

10.1.1 Firmware Gen 1

Revision	Date	Comments
2.1218	2024-07-12	PCIe interface optimizations Fixed first-level FIFO overflow Fixed rare bug causing PC freeze
2.1192	2023-06-14	Fixed bug related to packet polarity
2.1134	2021-12-10	Fixed TDC overtemp alarm issue
2.1126	2021-12-06	Fixed possible register read issues
2.1117	2021-06-23	Fixed register write issues
2.834	2017-12-05	Internal optimizations
2.797	2015-09-08	Hit sorting and packet generation issues fixed

10.1.2 Firmware Gen 2

Revision	Date	Comments
2.24192	2024-07-10	Fixed erroneous packet for very high trigger rates
2.24117	2024-04-26	Fixed bug related to PCIe readout
2.23060	2023-03-01	Fixed bug related to packet polarity
2.22341	2022-12-07	Minor bug fixes
2.22327	2022-11-18	Support for board revision 7

10.2 Driver & Applications

Revision	Date	Comments
1.10.7	2024-07-25	Support for TimeTagger4-10G (incl. calibration tool) Include baseline calibration Reduced supported range of `dc_offset` by 100 mV.
1.9.0	2023-07-10	Added quantization to timetagger4_param_info structure Code refactorization
1.8.3	2023-06-07	Minor bug fixes Code refactorization
1.8.2	2023-05-17	Added bounds and checks for various parameters
1.8.1	2023-05-09	Renamed autotrigger mode to continuous mode
1.8.0	2023-05-05	Added configurable input delay
1.7.0	2023-04-18	Board Revision 7 support TimeTagger4 : added autotrigger mode
1.4.5	2022-10-17	kernel driver v1.4.2 for xTDC4 only (fixes crash on Windows for Thunderbolt hot-plug)
1.4.4	2022-06-27	kernel driver v1.4.1
1.4.2	2021-07-28	Firmware updated ReadoutGUI added/updated User guide example added/updated
1.4.1	2019-11-11	x64 32 mode issues fixed
1.4.0	2019-06-04	Improved Windows 10 support
1.3.0	2019-01-23	Added Windows 10 support

10.3 User Guide

Revision	Date	Comments
1.10.4	2025-03-20	TimeTagger4: Updated Table 8.2.2
1.10.3	2025-03-18	xHPTDC8: Documented Firmware 2.1225 and Driver 1.5.0
1.10.2	2024-12-17	xHPTDC8: Updated Figure 2.3 xTDC4: Fixed formatting
1.10.1	2024-09-19	xHPTDC8: Fixes and additions to introduction xHPTDC8 and TimeTagger4: Documented max. readout rate for single channels Updated Figure 2.4
1.10.0	2024-08-14	TimeTagger4: Renamed 10G calibration tool Added Section “Memory Layout”
1.9.4	2024-07-30	Updated driver and firmware revision lists xHPTDC8: Updated user guide `example.cpp`

continued from previous page

Revision	Date	Comments
1.9.3	2024-07-16	xHPTDC8: Fix driver revision list
1.9.2	2024-07-09	xHPTDC8: Added LED documentation TimeTagger4 and xTDC4: Add overview figure of TBT and PCIe variant Fixed grammar
1.9.1	2024-07-02	xHPTDC8: Updated firmware list
1.9.0	2024-06-27	Added new driver revision TimeTagger4 and xTDC4: Added TBT variant TimeTagger4 and xTDC4: Added ordering information TimeTagger4 and xTDC4: Updated supported range for `dc_offset`
1.8.17	2024-06-20	xTDC4: Fixed API documentation
1.8.16	2024-06-20	TimeTagger4: Added documentation for 10G calibration tool xTDC4 and TimeTagger4: Added LED documentation xHPTDC8: Fixed default values for zero_channel Clarifications for TiGer block indices
1.8.15	2024-05-08	Fixed `auto_trigger` formula Updated oscillator characteristics xHPTDC8: Fixed mistakes in API xHPTDC8: Updated Code Examples
1.8.14	2024-03-27	Updated API Updated information on power consumption xHPTDC8: Extended chapter on gating
1.8.13	2024-01-18	xHPTDC8: Updated cover TimeTagger4: Updated feature list
1.8.12	2024-01-10	xHPTDC8: Updated driver revision history
1.8.11	2023-11-29	Reformatting Added latency between signal and Tiger output to Section 3.5 TimeTagger4: Updated table in Section 8.1.2 TimeTagger4: Clarifications in Features-list TimeTagger4: Added `ignore_empty_packets` API documentation xHPTDC8: Added default values for manager and configuration structs xHPTDC8: Fixed number of boards that can be synchronized from 8 to 6
1.8.10	2023-07-28	Changed extended range values to 0.429 s and 2.147 s, respectively. API clarifications.
1.8.9	2023-07-10	TimeTagger4 User Guide rework
1.8.8	2023-03-15	New TimeTagger4 variants -1.25G to -10G added
1.8.7	2022-11-24	Firmware revision notes updated
1.8.6	2022-11-23	Spelling and grammar corrections New example source code for xHPTDC8
1.8.5	2021-12-17	Clarifications related to TimeTagger4 configuration.
1.8.4	2021-12-08	Updated grouping structure in xHPTDC8 API
1.8.3	2021-07-28	Updated firmware revision history

continued from previous page

Revision	Date	Comments
1.8.2	2021-04-23	Added software trigger and _SYNC trigger sources for xHPTDC8 Corrected 3.3V power requirement for xHPTDC8 Changed types with fixed bit width to stdint.h for xHPTDC8 Added user flash functions for xHPTDC8
1.8.1	2021-04-09	Many corrections and updates to the xHPTDC8 API
1.8.0	2021-03-22	Added xHPTDC8 User Guide
1.7.0	2021-02-04	Combined User Guide for -1G and -2G Added characteristics for INL, DNL and Time Base Reordered sections for clarity Error corrections for rollovers, binsize and range Added figure 3.2 (TiGer matrix) Corrected board revision
1.6.0	2019-06-05	API clarifications

Erratum

We found undesired behavior for Gen 1 devices of the xTDC4.

If there are three or more edges close together (within 6.6 ns) and the user did only enable rising or falling edges but not both, some edges are reported with the wrong polarity. To trigger this behavior you need to violate the t_{DPfull} parameter of the board that states that for full resolution you may not have more than one pulse within a 5 ns interval.

If your configuration enables both edges all output data is correct. If you only need one type of edge (rising or falling) there are three simple workarounds:

a) update the Firmware of your Gen 1 device to svn`1192` or later.

b) enable both edges.

   All output words will be correct and your software can ignore all data that doesn't have the desired polarity.

c) enable only the desired edge polarity

   Ignore the polarity flag in the output data. You can trust that only edges with the desired polarity are output, even if the flag in the data word states the wrong polarity.