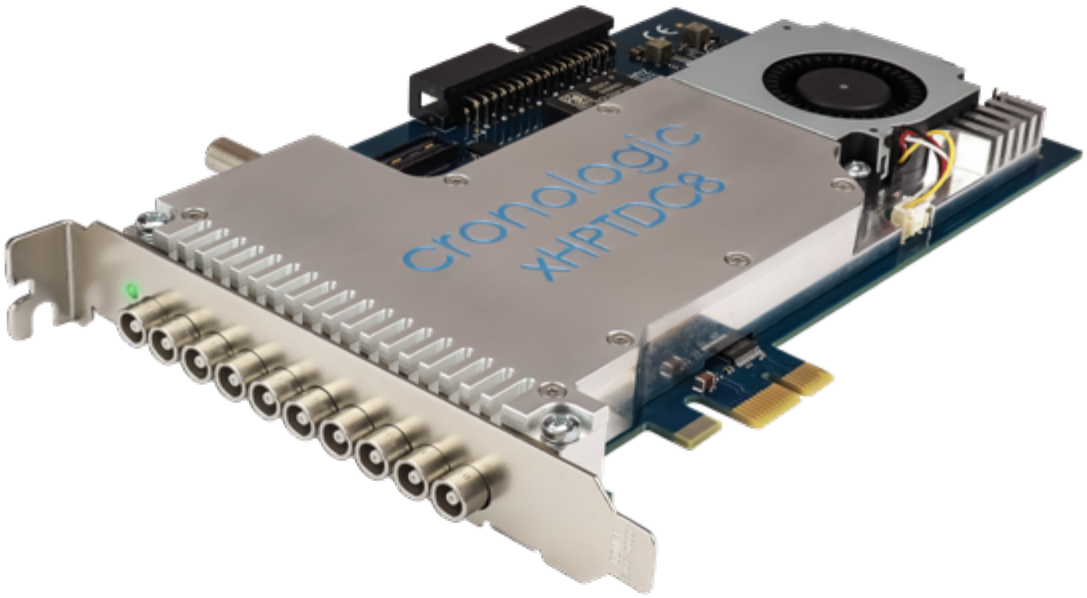
cronologic

xHPTDC8-PCIe

xHPTDC8-PCIe
User Guide



www.cronologic.de

Contents

1 Introduction

The xHPTDC8 is a streaming high-resolution time-to-digital converter. The time-of-arrival of leading or trailing edges of digital pulses are recorded in an infinite stream of timestamps.

A grouping mode is available that emulates common-start or common-stop behaviour.

1.1 Features

- 8-channel streaming TDC with 8 ps resolution
- Bin size: 13 ps
- Double-pulse resolution: 5 ns
- Dead time for readout: none
- L0 FIFO: 15 words/channel
- L1 FIFO: 512 words/channel
- L2 FIFO: 8000 words
- PCIe 1.1 x1 with 200 MB/s throughput
- auxiliary triggered ADC input

1.2 Applications

The xHPTDC8 can be used in all time measurement applications with up to eight channels with a single board or up to 64 channels when synchronising eight boards. For applications with four channels or less that do not require the flexibility of a streaming TDC, simpler products are available on our website www.cronologic.de.

The xHPTDC8 is well suited for the following applications:

- time-of-flight mass spectrometers (TOF-MS) with segmented detectors
- automated test equipment
- coincidence measurements
- quantum key distribution (QKD)
- time-correlated single-photon counting (TCSPC)
- synchronization of atomic clocks
- fluorescence lifetime imaging microscopy (FLIM)

2 Hardware

2.1 Installing the Board

The xHPTDC8 board can be installed in any PCIe-CEM slot with x1 or more lanes. Make sure the PC is powered off and the main power connector is disconnected while installing the board.

2.2 xHPTDC8 Inputs and Connectors

Figure 2.1 shows the location of the inputs on the slot bracket.

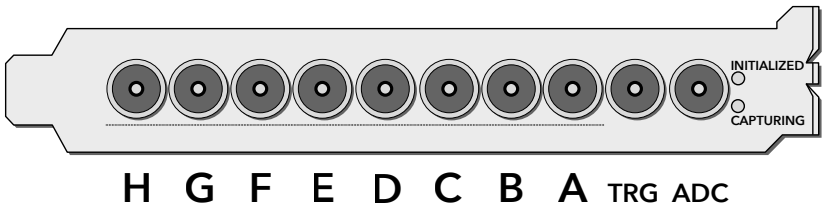


Figure 2.1 Input connectors of the xHPTDC8 on the PCIe bracket. Note, the TRG connector acts as a trigger only for the ADC channel, not the TDC channels (see Section 3.5).

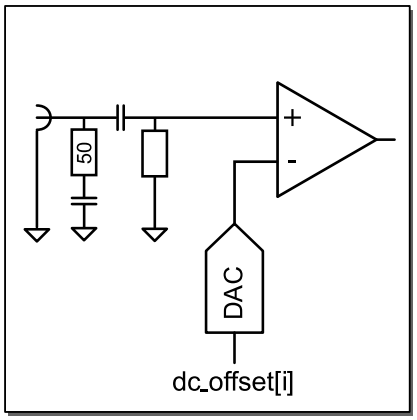


Figure 2.2 Input circuit for each of the input channels. Note, the TRG connector acts as a trigger only for the ADC channel, not the TDC channels (see Section 3.5).

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 2.2. The digital threshold for any

input can be 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. AC-coupled output pulses for automatic internal triggering and control of external devices can be generated using the TiGer timing pattern generator. See Section 3.3 for details on the TiGer. Furthermore three inter-board connectors can be found near

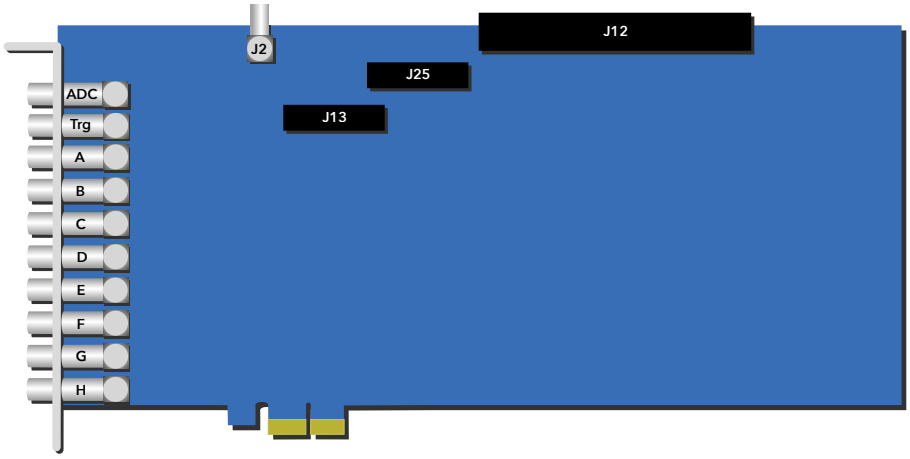


Figure 2.3 Schematic view of an xHPTDC8 Gen 1 board showing the inter-board connectors.

the top edge of the xHPTDC8 board, as displayed in Figure 2.3. 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. Connector J2 is a coax clock input that must receive a 10 MHz clock if multiple boards are used together as described in Section 2.3.

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 Synchronizing multiple boards

If more than eight TDC inputs are required, up to six boards can be synchronized within a system.

The xHPTDC8 API described in Chapter 4 manages up to six boards automatically and provides a single data stream that contains sorted hit data from all boards in chronological order. Channel A of each board is assigned channel number `board_index` × 10. The `board_index` is assigned to the boards in the order of the serial numbers starting at 0.

2.3.1 Connecting multiple boards

The boards must each receive a common 10 MHz clock on connector J2. The connector is inside the PC enclosure. Connectors J12 of all boards must be connected with a flat band cable with a terminator at each end. Cable and Terminator are available from cronologic. See Figure 2.4 for a wiring example.

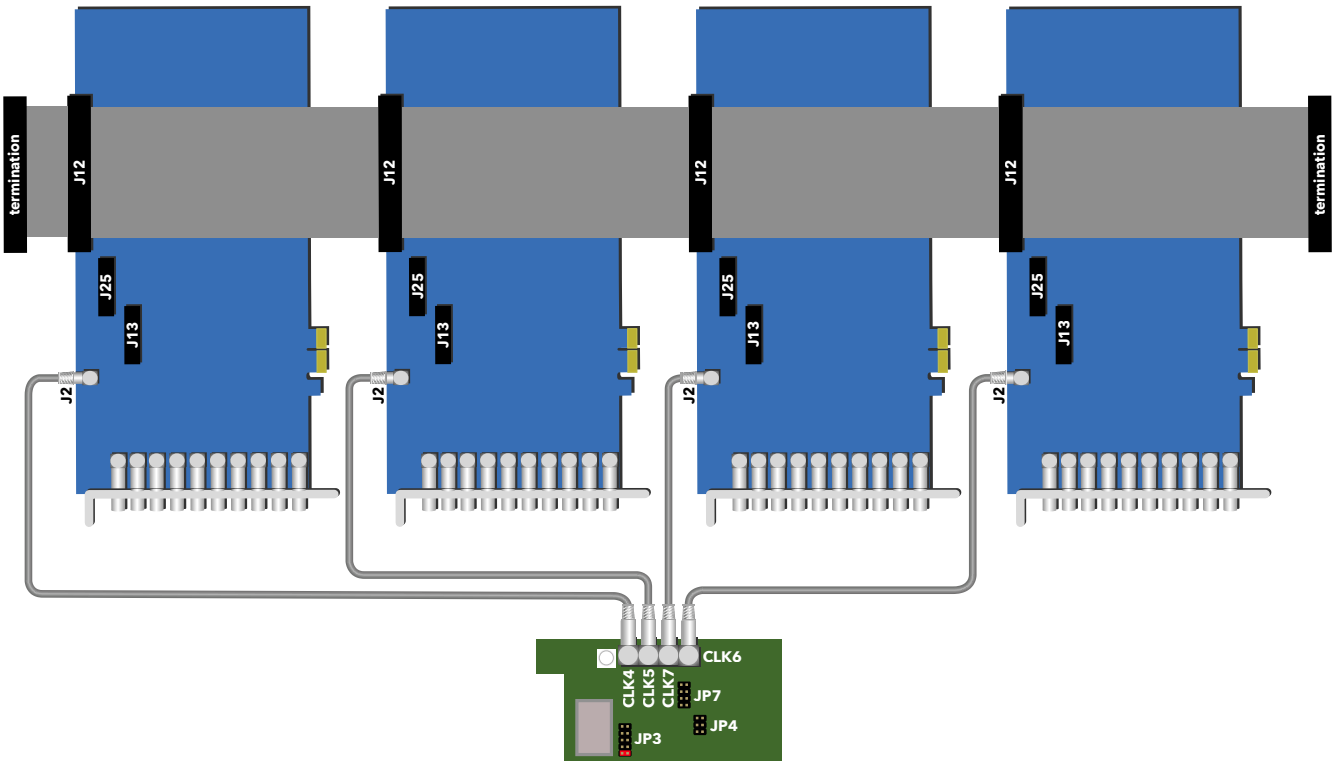


Figure 2.4 Synchronizing multiple boards with a ClockBox.

2.3.2 ClockBox

For systems of up to four boards, cronologic offers the ClockBox product that conveniently makes four clock signals available inside the PC enclosure. For use with the xHPTDC8, jumper JP3 of the ClockBox must be set as shown in Figure 2.5 in order to set the clock frequency to 10 MHz.

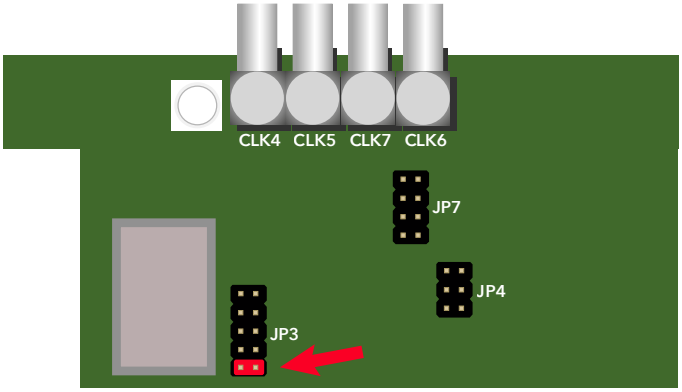


Figure 2.5 ClockBox jumper setting for 10 MHz.

2.3.3 Crates for multiple boards

Most PC mainboards don't have enough PCIe slots to support six xHPTDC8s. We offer an external enclosure called "Ndigo Crate" that uses PCIe-over-cable technology to extend the number of available slots in a system. The extension is fully transparent to the host system. There are no additional drivers required. Please see the product page at our website www.cronologic.de.

3 xHPTDC8 Functionality

The xHPTDC8 is a streaming time-to-digital converter. It records the timestamps of changes at the inputs A-H in an infinite stream. A flexible grouping mode is available that can emulate common-start or common-stop behaviour. See Section 3.1 for details.

For each channel, it can be selected individually whether rising or falling edges are recorded. It is not possible to record both edges of the same channel. The timestamps are recorded in integer multiples of a bin size of $5000/(3 \times 128) = 13.0208\overline{3}$ ps. There must be at least 5 ns between consecutive hits on the same input channel to be detected reliably. The xHPTDC8 records events without dead time at a readout rate of about 48 MHits/s.

3.1 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).

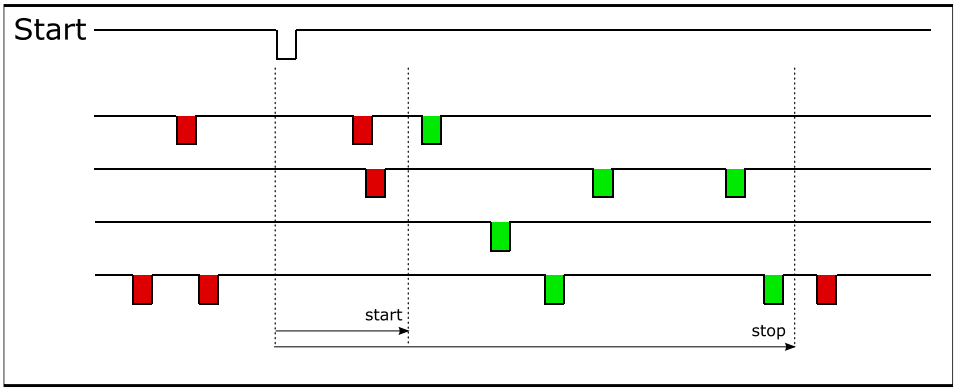


Figure 3.1 Acquired hits are merged to groups as explained in the text.

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 of that time window are discarded.

The start and stop values for grouping can also be negative. This allows to configure the xHPTDC8 for common-start, common-stop, or for groups that extend into both directions.

The values are configured in ps.

$$-2^{31} \leq start \leq stop \leq 2^{31} - 1$$

In single-board setups, it is recommended to also configure the gating blocks (see Page 29) to similar parameters as the grouping functionality. This prevents data from being read out that is discarded by the grouping code anyway. Please note that the grouping parameters are given in ps while the gating blocks are configured in cycles of the 150 MHz clock. For more information on the gating functionality, see Section 3.3.4.

In grouping mode, each call to `xHPTDC8_read_hits()` will return a group of timestamps relative to some trigger event. At the beginning of each group, an additional hit with channel number 255 is returned. This hit contains the absolute time of the group. The absolute time of the remaining hits can be obtained by adding this value to the relative time of each hit. Otherwise, the call returns all available timestamps as absolute timestamps counting upwards from `xHPTDC8_start_capture()`.

3.2 Auto-Triggering Function Generator

Some applications require internal periodic or random triggering. The xHPTDC8 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 of the 150 MHz.

The trigger can be used as a source for the TiGer unit (see Section 3.3) .

3.3 Timing Generator (TiGer)

Each digital LEMO-00 input can be used as an AC coupled trigger output. The TiGer functionality can be configured independently for each connector. See Section 4.5.5 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 and the ADC. Each TiGer can drive its output to its corresponding LEMO connector. This turns the connector into an output.

The TiGer outputs are AC coupled to the connector. They can be operated in one of the following modes:

`XHPTDC8_TIGER_OFF`
: No signal is output to the connector.

`XHPTDC8_TIGER_OUTPUT`
: In this mode the connector is output only. Pulses are unipolar with 2 V amplitude. Connected hardware must not drive any signals to connectors used as outputs, as doing so could damage both the xHPTDC8 and the external hardware. We recommend to only use short pulses to avoid undesirable baseline shift due to the AC coupling, but the device does not pose any restrictions on the duty cycle. This mode can be used as a clock output with a frequency of $75/N$ MHz for integer *N*.

`XHPTDC8_TIGER_BIDI`
: In this mode the TiGer creates unipolar pulses with 1 V amplitude. The connector can still be used as an input. Use short pulses to keep the probability of collision and the effect on the baseline low.

`XHPTDC8_TIGER_BIPOLAR`

In this mode the connector creates bipolar pulses with 1 V amplitude. The connector can still be used as an input. Pulses have no effect on the baseline offset. TiGer should be configured with *stop* = *start* for minimum width bipolar pulses of $2 \times 6.\overline{6}$ ns. The maximum bipolar pulse width is `XHPTDC8_TIGER_MAX_BIPOLAR_PULSE_LENGTH = 15`.

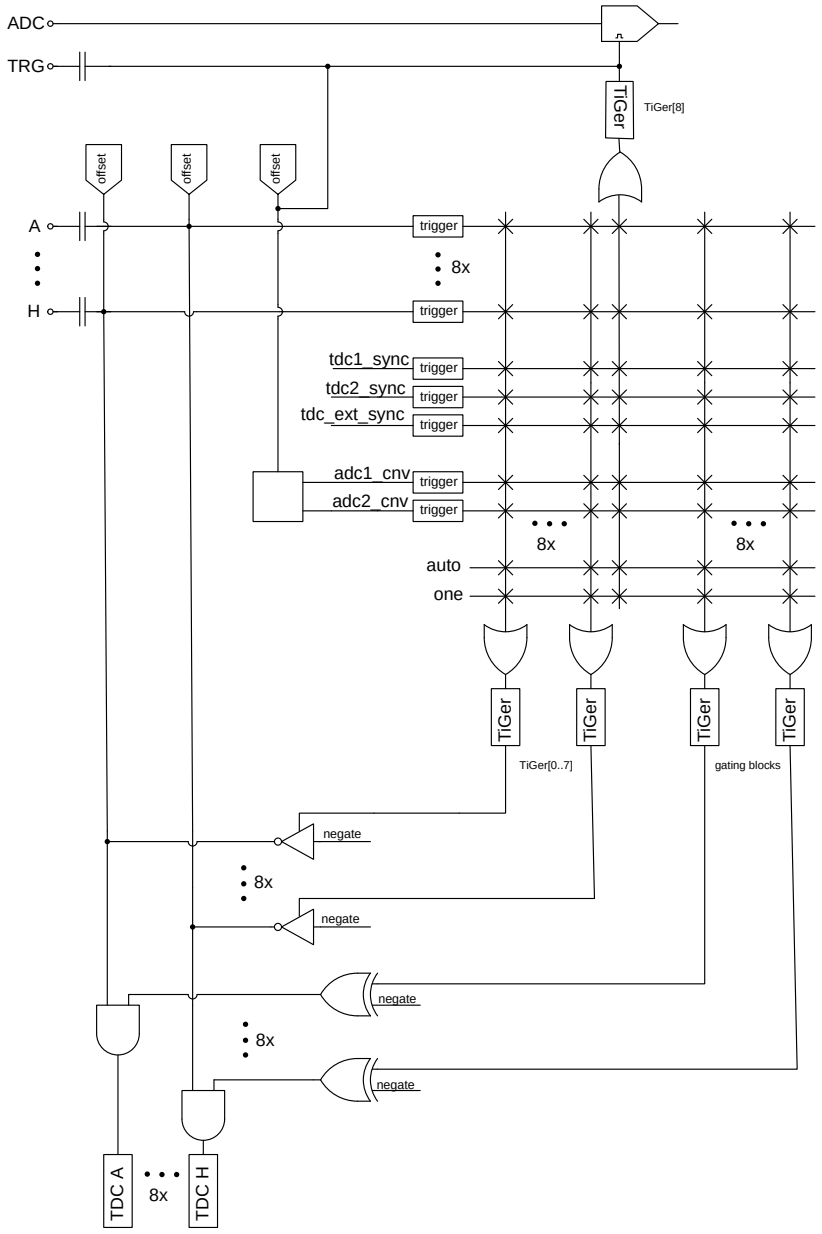


Figure 3.2 TiGer blocks can generate outputs that are also available on inputs.

3.3.1 Trigger Sources

Trigger sources for TiGer and gating blocks are configured by a bit mask. You can combine any number of trigger source with a bit-wise OR. The block will be trigger when any of the select trigger input is active.

`XHPTDC8_TRIGGER_SOURCE_NONE`
: Empty pattern that selects no trigger source.

`XHPTDC8_TRIGGER_SOURCE_A to _H`
: TDC LEMO inputs.

`XHPTDC8_TRIGGER_SOURCE_TDC1_SYNC`
: Same as `XHPTDC8_TRIGGER_SOURCE_TDC2_SYNC`.
Clock signal with 150/1024 MHz ≈ 146.5 kHz.

`XHPTDC8_TRIGGER_SOURCE_TDC_EXT_SYNC`
: Clock signal with 125 kHz.

`XHPTDC8_TRIGGER_SOURCE_AUTO`
: Periodic or random trigger pulses from the auto-trigger block.

`XHPTDC8_TRIGGER_SOURCE_ADC1_CONV`
: Same as `XHPTDC8_TRIGGER_SOURCE_ADC2_CONV`. When there is an ADC trigger pulse on the TRG connector, either of the two onboard ADCs is triggered in an unpredictable pattern. If the TRG input shall be used as a trigger, the trigger sources must contain both `ADC1_CNV` and `ADC2_CNV`.

`XHPTDC8_TRIGGER_SOURCE_SOFTWARE`
: Set for one clock cycle by a call to the driver function `xhptdc8_software_trigger()`.

`XHPTDC8_TRIGGER_SOURCE_ONE`
: Trigger that is active on every clock cycle.

3.3.2 TiGer Example: Generate 200 kHz Start Pulse

The AutoTrigger is used to generate a 200 kHz period. The TiGer generates a 200 kHz signal with 13.3 ns pulse width on LEMO-00 output A.

```
// generate an internal 200 kHz trigger
config.auto_trigger_period = 750;
config.auto_trigger_random_exponent = 0;

// setup TiGer
config.tiger_block[0].mode = XHPTDC8_TIGER_BIPOLAR;
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].sources = XHPTDC8_TRIGGER_SOURCE_AUTO;

// configure offset such that a 1V pulse can be detected by input A
config.dc_offset[0] = XHPTDC8_DC_OFFSET_NIM;
```

3.3.3 TiGer Example: Delayed Output from multiple sources

A trigger event on any channel B to D is used to generate a bipolar output pulse on channel A with configurable delay.

```
config.tiger_block[0].mode = XHPTDC8_TIGER_BIPOLAR;
config.tiger_block[0].start = 20;
config.tiger_block[0].stop = config.tiger_block[0].start + 1;
config.tiger_block[0].negate = 0;
// an event on any of the channels B - D starts the TiGer
config.tiger_block[0].sources = XHPTDC8_TRIGGER_SOURCE_B|XHPTDC8_TRIGGER_SOURCE_C|↩
    XHPTDC8_TRIGGER_SOURCE_D;
```

3.3.4 Triggering the ADC with the TiGer

There is a ninth TiGer that is connected to the trigger input (TRG, see Figure 2.1) of the ADC. See Section 3.5 for additional information on the ADC.

With retrigger enabled, the ADC TiGer can be used to periodically sample ADC data. The period should be no shorter than 300 ns or 45 TiGer clocks.

The ADC TiGer can also be used to sample voltages at a time relative to one of the TDC inputs. In this case, `stop` should be set to at least 45 to ensure that the sample period criterion is met even when pulses arrive in quick succession. A typical application would be to sample some slow control voltage once per start signal.

3.4 Gating

Each TDC channel has a second, identical TiGer block that functions as a gate as shown in Figure 3.2. In the driver configuration structure (see Section 4.5.6), these are accessible as `gating_block[channel]`.

Hits in a channel are only recorded while the gating block is open, which is configured by *Gate Start*, *Gate Stop*, *negate*, and *retrigger*. *Gate Start*, *Gate Stop*, and *retrigger* manipulate when the gate is *active* or *inactive*; *negate* manipulates if the gate is open while it is active or while it is inactive. If *negate* is false, the gate is open while its output is active. If *negate* is true, the gate is closed while its output is active.

If retriggering is enabled, the timer of the gate is reset when a second trigger event is recorded while the gate is active, i.e., the time while the gate is active is extended. If the second trigger event is recorded *before* the gate activates, the gate-timer is reset to zero, i.e., it will take the whole *Gate Start* time until the gate activates. If retriggering is disabled, all secondary trigger events will be ignored until the *Gate Stop* time is reached.

An overview of different combinations of *retrigger* and *negate* is shown in Figure 3.3.

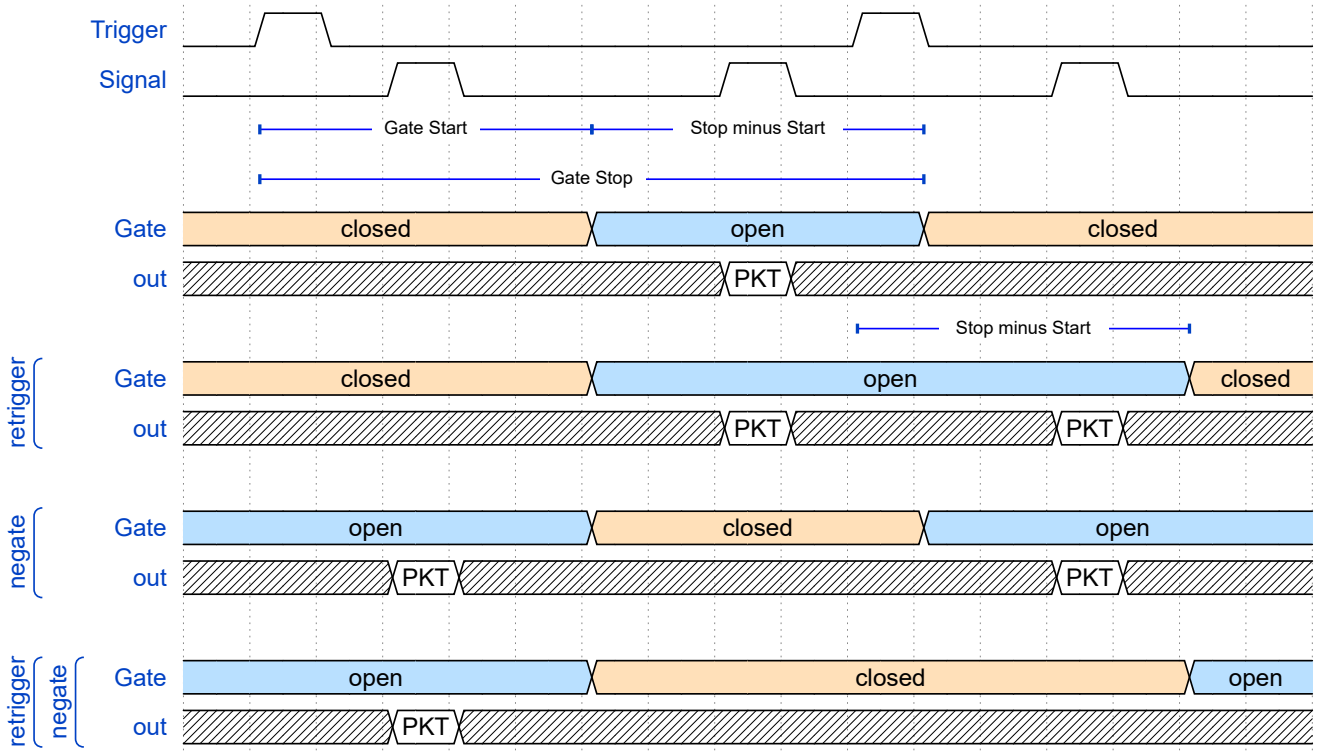


Figure 3.3 Principle of a gating block. Packets (PKT) are only recorded while the gate is open. *Gate Start*, *Gate Stop*, *retrigger*, and *negate* influence when that is the case.

Gating is a useful feature in setups where the trigger creates a lot of noise. A suitable configuration of the gating block can reduce the bandwidth and buffer usage significantly. Gating is performed before the L0 buffer. Grouping is performed in software after readout.

Note that the gating logic is not instant but takes about six to seven clock cycles due to board-internal signal processing. This means that there is a constant offset between the time an external trigger event is detected and the time the gating logic is enabled of up to about 50 ns. This offset does not exist if the (internal) auto-trigger functionality is utilized.

When setting up the gating range, it is recommended to conservatively choose the range (i.e., choose a bigger range than ultimately necessary). This is because at the edges of the gate, it is possible to create timestamps that do not correspond to a real signal edge due to how the gating logic is implemented: The gate stores the state of the input channel from the last time the gate was active and updates the state once the gate is re-enabled. This update may be interpreted as an edge, at which point a timestamp is created that does not necessarily correspond to a real edge of an input signal (see also Figure 3.4). However, timestamps of signals that are not spanning over the edge of the gate will always be correct.

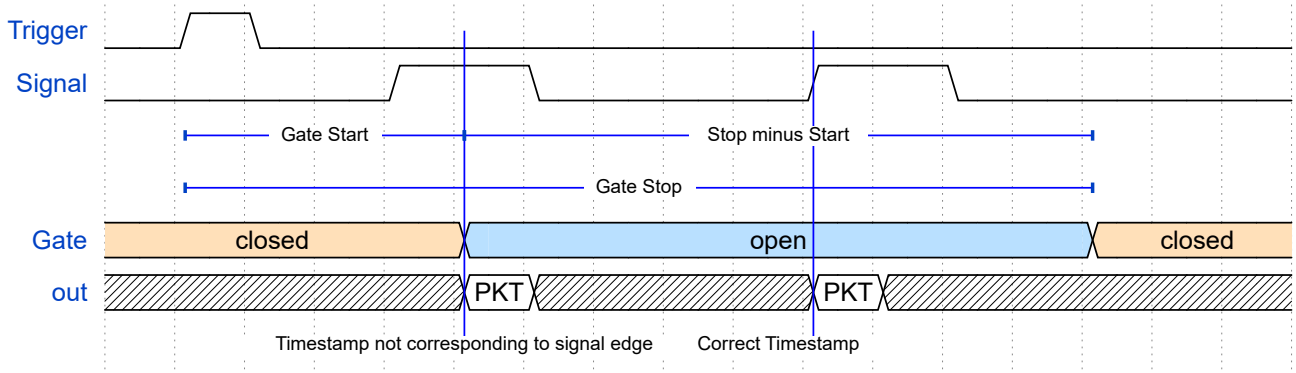


Figure 3.4 Example of a gating configuration that can lead to timestamps that do not correspond to a signal edge at the gate-edges. The gate keeps the last state from when it was active (low signal level in the example) and updates it once it re-activates (high signal level) which may be recorded as an edge.

3.5 Triggerable ADC

The xHPTDC8 is equipped with a triggerable ADC. Whenever there is a rising edge on the ADC trigger connector (TRG, see Figure 2.1), the voltage on the ADC input connector is sampled. The result is inserted as a packet with a timestamp and an ADC value into the readout data stream. The timing resolution of the timestamp is 833 ps.

TRG is also connected to the output of a TiGer block. This can be used to trigger the ADC periodical or relative to one of the TDC inputs as described in Section 3.3.4

The ADC triggers should be separated by at least 300 ns.

The width of the ADC input pulse should be larger than 13.2 ns and smaller than 35 ns.

There are two interleaved ADCs to ensure that there is always an ADC available even during read-out. This is exposed to the user both in the output data format and in the TiGer and Gating trigger sources. When using the ADC trigger as a trigger for Gating or TiGer, both trigger sources shall be set to the same value. During readout, the user shall not distinguish between data from the two ADCs unless advanced calibration is desired for the ADC data. In that case, the two ADCs should be treated separately.

4 Driver Programming API

The API is a DLL with C linkage.

The functions provided by the driver are declared in `xHPTDC8_interface.h` which must be included by your source code. You must tell your compiler to link with the file `xhptdc8_driver.lib` or the corresponding 64 bit version `xhptdc8_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\xHPTDC8\driver`. A coding example can be found on github.com/cronologic-de/xtdc_babel.

There exist an open-source community project that intends to provide some convenient extensions to the driver, code examples, and wrappers to make the driver usable with various programming languages like Python and LabView. The project is hosted at https://github.com/cronologic-de/xhptdc8_babel.

4.1 Constants

`#define XHPTDC8_MANAGER_DEVICES_MAX 8`
The maximum number of boards supported by the device manager.

`#define XHPTDC8_TDC_CHANNEL_COUNT 8`
The number of TDC input channels.

`#define XHPTDC8_GATE_COUNT 8`
The number of gating blocks. One for each TDC input.

`#define XHPTDC8_TIGER_COUNT 9`
The number of timing generators. One for each TDC input and one for the ADC trigger.

`#define XHPTDC8_TRIGGER_COUNT 16`
The number of potential trigger sources for the timing generators. One for each TDC input plus some specials. See Section 4.5.4 for details.

`#define XHPTDC8_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 `xHPTDC8_interface.h`

4.2 Driver Information

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

`int xhptdc8_get_driver_revision()`

Returns the driver version, same format as `xhptdc8_static_info.driver_revision`. This function does not require an xHPTDC8 board to be present.

`const char* xhptdc8_get_driver_revision_str()`
Returns the driver version including SVN build revision as a string.

`int xhptdc8_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 XHPTDC8_OK = 0`, the error message will contain what went wrong. E.g., 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 init parameters and change some values. E.g., choose one of multiple cards by the index or use a larger buffer.

`int xhptdc8_get_default_init_parameters(xhptdc8_manager_init_parameters *init)`
Sets up the standard parameters. Gets a set of default parameters for `xhptdc8_init()`. This must always be used to initialize the `xhptdc8_manager_init_parameters` structure before modifying it and passing it to `xhptdc8_init`.

`int xhptdc8_init(xhptdc8_manager_init_parameters *params)`
Opens and initializes the xHPTDC8 boards in the system.

If the return value does not equal `#define XHPTDC8_OK = 0` the device initialization failed. The parameter `params` is a pointer to a structure of type `xhptdc8_manager_init_parameters` that must be completely initialized by `get_default_init_parameters()`.

`int xhptdc8_close()`
Closes the devices, releasing all resources.

4.3.1 Structure xhptdc8_manager_init_parameters

`int version = XHPTDC8_API_VERSION;`
The version number. Must be set to `XHPTDC8_API_VERSION`.

`uint64_t buffer_size = XHPTDC_DEFAULT_BUFFER_SIZE; // 0x1000000 (16 MB)`
The minimum size of the DMA buffer.
If set to `0` the default size of 16 MByte is used.

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

`int device_type = CRONO_DEVICE_XHPTDC8;`
A constant for the different devices of cronologic `CRONO_DEVICE_*`.

Initialized by `xhptdc8_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 = 250;`
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.

`crono_bool_t multiboard = false;`
Set if multiple devices shall be synchronized. Also sets the clock source to external.

`crono_bool_t use_ext_clock = false;`
If set use the external 10 MHz reference on J2, otherwise the internal clock source is used. When `multiboard` is set, this parameter is ignored and the external clock is used.

`crono_bool_t ignore_calibration = false;`
Leave at `false` to use device calibration data.

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.

The information is provided on a per board basis. The parameter `index` selects which board is queried.

`int xhptdc8_get_device_type(int index)`
Returns the type of the device as `CRONO_DEVICE_XHPTDC8`.

`const char* xhptdc8_get_last_error_message(int index)`
Returns most recent error message. If index is negative the last error message from the `xhptdc8_manager` is returned. Otherwise, the last error message of the selected board is returned.

`int xhptdc8_get_fast_info(int index, xhptdc8_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 xhptdc8_get_param_info(int index, xhptdc8_param_info *info)`
Returns configuration changes.

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

`int` **xhptdc8_get_static_info(int index, xhptdc8_static_info *info)**
Contains static information.

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

`int` **xhptdc8_get_pcie_info(int index, crono_pcie_info *pcie_info)**
Read PCIe information.

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

`int` **xhptdc8_clear_pcie_errors(int index, 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
```

`int` **xhptdc8_get_temperature_info(int index, xhptdc8_temperature_info *info)**
Get temperature measurements from multiple sources on the board.

`int` **xhptdc8_get_clock_info(int index, xhptdc8_clock_info *info)**
Get information on clocking configuration and status.

`const char*` **xhptdc8_device_state_to _str(int state)**
Convert the state value from `xhptdc8_fast_info.state` into a human-readable string.

4.4.2 Structure xhptdc8_static_info

This structure contains information about the board that does not change during run time. It is provided by the function `xhptdc8_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.

All xHPTDC8 boards in the system are numbered in the order of their serial numbers starting at zero. Channel A of a board has channel number *index* × 10.

`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[2]`**
16 bit factory ID for each of the TDC chips.

`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.

`uint32_t` **`flash_serial_high`**
`uint32_t` **`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 xHPTDC8.

`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 xhptdc8_param_info

This structure contains configuration changes provided by `xhptdc8_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` **`channels`**
Number of TDC channels of the board.

Fixed at 8.

`int` **`channel_mask`**
Bit assignment of each enabled input channel.

Bit $0 \leq n < 8$ is set if channel n is enabled.

`int64_t` **`total_buffer`**
The total amount of DMA buffer in bytes.

4.4.4 Structure xhptdc8_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` **`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 xHPTDC8.

`int` **`pcie_max_payload`**
Maximum size in bytes for one PCIe transaction. Depends on system configuration.

`int` **`state`**
Current state of the xHPTDC8.

```
const static int XHPTDC8_DEVICE_STATE_CREATED      0
const static int XHPTDC8_DEVICE_STATE_INITIALIZED  1
const static int XHPTDC8_DEVICE_STATE_CONFIGURED   2
const static int XHPTDC8_DEVICE_STATE_CAPTURING    3
const static int XHPTDC8_DEVICE_STATE_PAUSED       4
const static int XHPTDC8_DEVICE_STATE_CLOSED       5
```

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
```

4.4.6 Structure xhptdc8_temperature_info

This structure provide the values of temperature measurements of various chips on 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.

`float` tdc[2]
Temperature for each of the TDC chips in °C.

4.4.7 Structure xhptdc8_clock_info

This structure provides information about the clock network of the device.

`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.

`crono_bool_t` cdce_locked
Set if the jitter cleaning PLL clock synthesizer achieved lock.

`int` cdce_version
Version information from the CDCE62005 clock synthesizer.

`crono_bool_t` **use_ext_clock**
: Source for the clock synthesizer is usually the 10 MHz onboard oscillator. During initialization, alternatively an external clock on J2 can be selected. When multiple boards are synchronized all board use a common external clock. See section 2.3 for details.

`crono_bool_t` **fpga_locked**
: Set if the FPGA datapath PLL achieved lock.

4.5 Configuration

All xHPTDC8 boards in the system are configured by a single configuration structure which in turn contains sub structures that configure the individual boards. 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` **xhptdc8_configure(**`xhptdc8_manager_configuration` ***config)**
: Configures the `xhptdc8_manager`.

`int` **xhptdc8_get_current_configuration(**`xhptdc8_manager_configuration` ***config)**
: Gets current configuration. Copies the current configuration to the specified config pointer.

`int` **xhptdc8_get_default_configuration(**`xhptdc8_manager_configuration` ***config)**
: Gets default configuration. Copies the default configuration to the specified config pointer.

4.5.1 YAML config files

There exist a community maintained utility library for the xHPTDC8 that contains a convenience function that can modify configuration structures from a YAML config string. This can significantly shorten code required to setup the TDC.

See github.com/cronologic-de/xhptdc8_babel/tree/main/util for details.

4.5.2 Structure xhptdc8_manager_configuration

`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.

`xhptdc8_device_configuration` **device_configs[XHPTDC8_DEVICES_MAX]**
: A structure with the configuration for an individual xHPTDC8 board as defined in section 4.5.3. Use the function `xhptdc8_count_devices()` to query how many entries contain valid information. See Section 4.2 for details on the function.

`xhptdc8_grouping_configuration` **grouping**
: Structure with the parameters for grouping.

  See Section 4.5.8 for the definition of the structure and Section 3.1 for more information on grouping.

`int64_t *bin_to_ps(int64_t)`
Reserved for future use.

4.5.3 Structure xhptdc8_device_configuration

This is the structure containing the configuration information. 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 auto_trigger_period = 300000000;`
`int auto_trigger_random_exponent = 0;`
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.

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

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.

`double threshold[XHPTDC8_TDC_CHANNEL_COUNT] = -0.35;`
Set the threshold voltage for the input channels A …H (see Figure 4.1).

The supported range is −1.32 V to 1.18 V. This should be close to 50% of the height of the input pulse. Examples for various signaling standards are defined as follows:

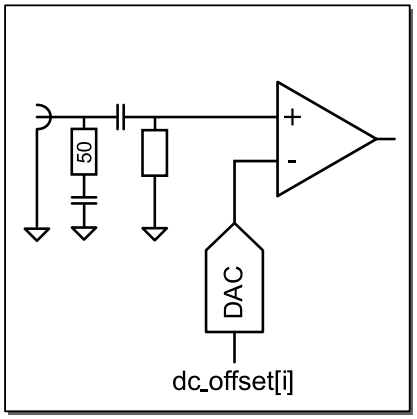


Figure 4.1 Input circuit for each of the input channels.

```
#define XHPTDC8_THRESHOLD_P_NIM        +0.35
#define XHPTDC8_THRESHOLD_P_CMOS       +1.18
#define XHPTDC8_THRESHOLD_P_LVCMOS_33  +1.18
#define XHPTDC8_THRESHOLD_P_LVCMOS_25  +1.18
#define XHPTDC8_THRESHOLD_P_LVCMOS_18  +0.90
#define XHPTDC8_THRESHOLD_P_TTL        +1.18
#define XHPTDC8_THRESHOLD_P_LVTTL_33   +1.18
#define XHPTDC8_THRESHOLD_P_LVTTL_25   +1.18
#define XHPTDC8_THRESHOLD_P_SSTL_3     +1.18
#define XHPTDC8_THRESHOLD_P_SSTL_2     +1.18
#define XHPTDC8_THRESHOLD_N_NIM        -0.35
#define XHPTDC8_THRESHOLD_N_CMOS       -1.32
#define XHPTDC8_THRESHOLD_N_LVCMOS_33  -1.32
#define XHPTDC8_THRESHOLD_N_LVCMOS_25  -1.25
#define XHPTDC8_THRESHOLD_N_LVCMOS_18  -0.90
#define XHPTDC8_THRESHOLD_N_TTL        -1.32
#define XHPTDC8_THRESHOLD_N_LVTTL_33   -1.32
#define XHPTDC8_THRESHOLD_N_LVTTL_25   -1.25
#define XHPTDC8_THRESHOLD_N_SSTL_3     -1.32
#define XHPTDC8_THRESHOLD_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.

`xhptdc8_trigger` **`trigger[XHPTDC8_TRIGGER_COUNT]`**
Configuration of the polarity of the external trigger sources (see Section 4.5.4). These are used as inputs for the TiGer blocks and as inputs to the time measurement unit.

`xhptdc8_tiger_block` **`tiger_block[XHPTDC8_TIGER_COUNT]`**
Configuration of the timing generators (TiGer, see Section 4.5.5).

`xhptdc8_tiger_block` **`gating_block[XHPTDC8_GATE_COUNT]`**
Configuration of the gating blocks.

`xhptdc8_channel` **`channel[XHPTDC8_TDC_CHANNEL_COUNT]`**
Configuration of the TDC channels.

`xhptdc8_channel` **`adc_channel`**
Configuration of the ADC channel.

`crono_bool_t` **`skip_alignment`** `= false;`
If set, the phase of the two TDC chips is not realigned when capturing is restarted.

`int` **`alignment_source`** `= 1;`
Define a signal source that is used for phase alignment. Should usually be left unchanged.

```
#define XHPTDC8_ALIGN_TIGER     0
#define XHPTDC8_ALIGN_PIN       1
#define XHPTDC8_ALIGN_RESERVED  2
```

`int` **`alignment_off_state`** `= 0;`
Select TDC alignment pin state when not in use.

`0`	GND
`1`	VCCIO
`2`	high-Z

4.5.4 Structure xhptdc8_trigger

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

`crono_bool_t` **`falling`** `= true;`
`crono_bool_t` **`rising`** `= false;`
Select for which edges a trigger event is created inside the FPGA. While the TDC can only measure either rising or falling edges, the gating blocks and the TiGer are more flexible. Set the corresponding flag for one of the edges or both edges when using the input with a TiGer or gating block.

4.5.5 Structure xhptdc8_tiger_block

See Section 3.3 for additional information.

`int` **`mode`** `= 0;`

Enables the desired mode of operation for the tiger.

`#define XHPTDC8_TIGER _OFF`	`0`	No operation
`#define XHPTDC8_TIGER _OUTPUT`	`1`	Output is driven with 2 V amplitude. There must be no input connected
`#define XHPTDC8_TIGER _BIDI`	`2`	Output is driven with 1 V amplitude. Pulse rate should be low.
`#define XHPTDC8_TIGER _BIPOLAR`	`3`	Output is driven with 1 V bidirectional pulses. $start = stop - 1$

The gating blocks are only used internally and produce no pulses accessible to the user. Gating blocks interpret any value that is not `0` as enable.

`#define XHPTDC8_GATE _OFF`	`0`	No gating, all hits are captured.
`#define XHPTDC8_GATE _ON`	`1`	No hits are captured while the gate is inactive.

`crono_bool_t` **`negate`** `= false;`

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

For gating blocks, a value of `false` enables inputs between *start* and *stop*, a value of `true` blocks outputs inside that interval. The TiGer creates a high pulse from *start* to *stop* unless negated.

`crono_bool_t` **`retrigger`** `= false;`

Enables retrigger 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`** `= true;`

Not implemented.

`uint32_t` **`start`** `= 0;`
`uint32_t` **`stop`** `= 1000;`

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

For gating blocks, there is a constant offset of about six to seven cycles between *start/stop* and the time an external input signal is detected (see also Section 3.3.4).

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

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

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`** `= 0x00000001;`

A bit mask with a bit set for all trigger sources that can trigger this TiGer block. Default is `XHPTDC8_TRIGGER_SOURCE_A`.

```
#define XHPTDC8_TRIGGER_SOURCE_NONE          0x00000000
#define XHPTDC8_TRIGGER_SOURCE_A             0x00000001
#define XHPTDC8_TRIGGER_SOURCE_B             0x00000002
#define XHPTDC8_TRIGGER_SOURCE_C             0x00000004
#define XHPTDC8_TRIGGER_SOURCE_D             0x00000008
#define XHPTDC8_TRIGGER_SOURCE_E             0x00000010
#define XHPTDC8_TRIGGER_SOURCE_F             0x00000020
#define XHPTDC8_TRIGGER_SOURCE_G             0x00000040
#define XHPTDC8_TRIGGER_SOURCE_H             0x00000080
#define XHPTDC8_TRIGGER_SOURCE_TDC1_SYNC     0x00000100
#define XHPTDC8_TRIGGER_SOURCE_TDC2_SYNC     0x00000200
#define XHPTDC8_TRIGGER_SOURCE_TDC_EXT_SYNC  0x00000400
#define XHPTDC8_TRIGGER_SOURCE_ADC1_CONV     0x00000800
#define XHPTDC8_TRIGGER_SOURCE_ADC2_CONV     0x00001000
#define XHPTDC8_TRIGGER_SOURCE_SOFTWARE      0x00002000
#define XHPTDC8_TRIGGER_SOURCE_AUTO          0x00004000
#define XHPTDC8_TRIGGER_SOURCE_ONE           0x00008000
```

4.5.6 Structure xhptdc8_channel

Contains TDC channel settings.

`crono_bool_t enable = false;`
Enable the TDC channel.

`crono_bool_t rising = false;`
Select which edge of the signal is measured by the TDC. The TiGer and gating blocks use a separate configuration that allows to use both edges simultaneously on each input (see Section 4.5.4).

4.5.7 Structure adc_channel

This structure configures the ADC input and the corresponding trigger input. See section 3.5.

`crono_bool_t enable = false;`
Enable acquisition of ADC data.

`crono_bool_t watchdog_readout = false;`
Include periodic ADC measurements in the output data. Watchdog measurements do not inhibit ADC triggers occurring at the same time.

`int` **`watchdog_interval`** `= 6144;`
Number of 150-MHz clock cycles within one watchdog period.

$$100 \leq \texttt{watchdog\_interval} \leq 7500$$

`double` **`trigger_threshold`** `= -0.35;`
Threshold voltage for the TRG input. See the description for the `channel.trigger_threshold` in Section 4.5.3.

4.5.8 Structure xhptdc8_grouping_configuration

This structure configures the behaviour of the grouping functionality (see Section 3.1).

In this structure intervals are always provided in picoseconds, independently of the bin size of the TDC.

`crono_bool_t` **`enabled`** `= false;`
Enable grouping.

`int` **`trigger_channel`** `= 0;`
Channel number that is used to trigger the creation of a group.

`uint64_t` **`trigger_channel_bitmask`** `= 0ull;`
Use this to define additional trigger channels. There is an OR-disjuction with the `trigger_channel`.

`int` **`zero_channel`** `= 0;`
Optionally a different channel can be used to calculate the relative timestamps in a group. This is disabled per default by setting this parameter to `-1`.

`int64_t` **`zero_channel_offset`** `= 0;`
This offset in picoseconds is added to relative timestamps within a group.

`int64_t` **`range_start`** `= -1500;`
Start of group range relative to the `trigger_channel`.

`int64_t` **`range_stop`** `= 1500;`
End of group range relative to the `trigger_channel`.

Values in the interval from `range_start` to `range_stop` are included in the group. Either or both values can be negative to create common-stop behaviour.

$$-2^{63} \leq \texttt{range\_start} \leq \texttt{range\_stop} < 2^{63}$$

`int64_t` **`trigger_deadtime`** `= 0;`
After a trigger was detected additional triggers will be suppressed for this interval. Must not be negative.

`uint64_t` **`window_hit_channels`** `= 0ull;`
Set a bitmask of channels. A group is only created, if there is at least one hit in the windows defined by `window_start` and `window_stop`. Usage is equivalent to `trigger_channel_bitmask`.

`int64_t` **`window_start`** `= 0;`
`int64_t` **`window_stop`** `= grouping.window_start + 1;`

$$-2^{63} \leq \texttt{window\_start} \leq \texttt{window\_stop} < 2^{63}$$

`int veto_mode = 0;`
A window defined by `veto_start` and `veto_stop` can be used to suppress hits. The functionality is very similar to the gating blocks but is defined in software. While gating blocks can only work locally on the information available within each board the veto function is applied globally accross all boards. This feature cannot be used to improve FIFO usage or PCIe bandwidth usage.

Either data inside or outside the veto window can be suppressed.

```
#define XHPTDC8_GROUPING_VETO_OFF      0
#define XHPTDC8_GROUPING_VETO_INSIDE   1
#define XHPTDC8_GROUPING_VETO_OUTSIDE  2
```

`int64_t veto_start = 0;`
`int64_t veto_stop = 0;`

$$-2^{63} \leq \texttt{veto\_start} \leq \texttt{veto\_stop} < 2^{63}$$

`uint64_t veto_active_channels = 0xffffffffffffffffull;`
If veto is enabled, veto filtering is active for channels defined by a channel bitmask. As default, filtering is active for all channels.

`crono_bool_t veto_relative_zero = 0;`
If set, the veto window is defined relative to the zero channel. Otherwise, the window is defined relative to the trigger.

`crono_bool_t ignore_empty_events = 0;`
Discard groups which contained only a trigger signal.

`crono_bool_t overlap = 0;`
Unsupported, must remain `false`.

4.6 User Data Storage

There is a 64 kByte flash memory on each board that users can utilize to store any type of data. A typical use case would be calibration data for the xHPTDC8 or the detectors that the device is connected to. Also serial numbers of instruments built with the xHPTDC8 can be stored here. Write operations always erase the whole memory block.

`#define XHPTDC8_USER_FLASH_SIZE 0x10000`
The size of the flash memory in bytes.

`int xhptdc8_read_user_flash(int index, uint8_t* flash_data, uint32_t size)`
`int xhptdc8_write_user_flash(int index, uint8_t* flash_data, uint32_t size)`
Read from or write to the user flash of a board identified by `index`. `flash_data` must point to a buffer allocated by the user. `size` must specify the size of that buffer in bytes. We recommend to always allocate a buffer of the size of the flash memory given by `XHPTDC8_FLASH_SIZE` to clarify that always the full buffer is overwritten.

5 Run Time Control

5.1 Controlling the State of the Driver

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

`int` **`xhptdc8_start_capture()`**
Start data acquisition.

`int` **`xhptdc8_pause_capture()`**
Pause a started data acquisition. Pause and continue have less overhead than start and stop but don't allow for a configuration change.

`int` **`xhptdc8_continue_capture()`**
Call this to resume data acquisition after a call to `xhptdc8_pause_capture()`. Pause and continue have less overhead than start and stop but don't allow for a configuration change.

`int` **`xhptdc8_stop_capture()`**
Stop data acquisition.

`int` **`xhptdc8_start_tiger(int index)`**
`int` **`xhptdc8_stop_tiger(int index)`**
Start and stop the timing generator of an individual board. This can be done independently of the state of the data acquisition.

`int` **`xhptdc8_software_trigger(int index)`**
Sets the software trigger for one clock cycle. This can be configured for the TiGer and for the gating blocks as trigger source `XHPTDC8_TRIGGER_SOURCE_SOFTWARE`.

5.2 Readout

`int` **`xhptdc8_read_hits(TDCHit *hit_buf, size_t size)`**
Read a multitude of hits into a buffer provided by the user. Returns the number of read hits.

If grouping is enabled a single group is read. If the group is too large for the buffer the remaining hits of the group are discarded.

If grouping is disabled, all available data is read up to the size of the buffer.

The method always returns immediately. If no hits are read, it is beneficial to call `sleep()` or yield the CPU to another process instead of trying again immediately.

Make sure to set `size` to the number of elements that fit into the buffer.

This function is not thread-safe. If you want to process the read data in multiple threads the data needs to be copied to a separate buffer for each thread.

`int` **`xhptdc8_get_current_timestamp(int index, int64_t *timestamp)`**
Return current internal timestamp counter value of the selected xHPTDC8 in picoseconds.

6 Output Data Format

For each measured edge, the xHPTDC8 creates a 12-byte data structure TDCHit that contains a 64-bit timestamp in picoseconds and three fields with additional information.

6.1 Structure TDCHit

`int64_t time`
The timestamp of the hit in picoseconds.

If grouping is disabled the timestamps are continuously counting up from the call to `xhptdc8_start_capture()`.

If grouping is enabled the timestamps are relative to the trigger or the separate zero reference of the group. The first TDCHit of a group has channel number 255 and provides the absolute time of the group. The absolute time of each of the hits can be obtained by adding this value to each of the relative timestamps.

`uint8_t channel`
For the first board in the system, this is `0` to `7` for the TDC channels A to H, or `8` to `9` for ADC data. Data from channels 8 and 9 should usually be treated as data from the same channel. For the other boards, the channel number is incremented by `board_id` × 10. In grouping mode, the first hit of each group has channel number 255 and contains the absolute time of the group.

`uint8_t type`
Additional information on the type of hit recorded (see Section 6.1.1).

`uint16_t bin`
For ADC hits this contains the sampled voltage. For TDC hits the content is undefined.

6.1.1 TDCHit Types

Type information for TDC measurements

If the hit is a TDC measurement on channels A to H the following flags are defined for the `type` field of the TDCHit structure:

`#define XHPTDC8_TDCHIT_TYPE_RISING 0x01`
Rising edge

`#define XHPTDC8_TDCHIT_TYPE_ERROR 0x02`
any type of error

`#define XHPTDC8_TDCHIT_TYPE_ERROR_TIMESTAMP_LOST 0x04`
Hits missing due to L1 FIFO overflow

`#define XHPTDC8_TDCHIT_TYPE_ERROR_ROLLOVER_LOST 0x08`
Invalid timestamp due to internal error

`#define` **`XHPTDC8_TDCHIT_TYPE_ERROR_PACKETS_LOST 0x10`**
Hits missing due to a lost DMA packet

`#define` **`XHPTDC8_TDCHIT_TYPE_ERROR_SHORTENED 0x20`**
Hits missing due to a shortened DMA packet

`#define` **`XHPTDC8_TDCHIT_TYPE_ERROR_DMA_FIFO_FULL 0x40`**
Hits missing due to L2 FIFO overflow

`#define` **`XHPTDC8_TDCHIT_TYPE_ERROR_HOST_BUFFER_FULL 0x80`**
Hits missing due to host buffer overflow

If hits are missing the error flag is set on the next hit from the same board that is read out.

Type information for ADC measurements

If the hit is an ADC measurement on channels 8 or 9, the following flags are defined for the `type` field of the TDCHit structure:

`#define` **`XHPTDC8_TDCHIT_TYPE_ADC_INTERNAL 0x01`**
ADC measurement triggered by internal strobe

`#define` **`XHPTDC8_TDCHIT_TYPE_ADC_ERROR 0x02`**
any type of error

`#define` **`XHPTDC8_TDCHIT_TYPE_ADC_ERROR_INVALID_TRIGGER 0x08`**
TRG input violated timing requirements. Data may be corrupted

`#define` **`XHPTDC8_TDCHIT_TYPE_ADC_ERROR_DATA_LOST 0x10`**
ADC measurement missing due to overflow of any buffer

If hits are missing the error flag is set on the next hit from the same board that is read out.

7 Code Example

The following C++ source code shows how to initializes a an xHPTDC8 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 this file. You can open it in an external viewer or save it to disk by clicking on it.

The example code is managed as open-source on GitHub at https://github.com/cronologic-de/xhptdc8_babel. The repository contains a complete project for Microsoft Visual Studio that you can use to compile the example. Examples for more programming languages such as Golang, Rust, Python, and LabView will be added to the repository over time.

At the time this document is written, the following source code can be found in the repository:

- dummy library to be able to develop code for the xHPTDC8 without a physical board being present
- utility library to configure the device from YAML strings or YAML files
- command line info tool to list information about all xHPTDC8 boards in the system. This tool is written in Go.

```
// xhptdc8_user_guide_example.cpp : Example application for the xHPTDC8

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include "crono_interface.h"
#include "xHPTDC8_interface.h"

typedef unsigned int uint32;
typedef unsigned __int64 uint64;
int exit_on_fail(int status, const char* message);
const int MAX_TRYS_TO_READ_HITS = 1000;

// create a manager object that provides access to all xHPTDC8 in the system
int initialize_xhptdc8(int buffer_size) {
  // prepare initialization
  xhptdc8_manager_init_parameters params;

  xhptdc8_get_default_init_parameters(&params);
  params.buffer_size = buffer_size;

  int error_code;
  char* error_msg = NULL;
  error_code = xhptdc8_init(&params);
  exit_on_fail(error_code, error_msg);
  return error_code;
}

```

```
int get_device_count() {
  int error_code;
  char* error_msg;

  int device_count = xhptdc8_count_devices(&error_code, (const char**)&error_msg);
  exit_on_fail(error_code, error_msg);
  return device_count;
}

int configure_xhptdc8(int device_count) {
  xhptdc8_manager_configuration* mgr_cfg = new xhptdc8_manager_configuration;
  xhptdc8_get_default_configuration(mgr_cfg);
  int general_offset = 50, epsilon = 4;

  // configure all devices with an identical configuration
  for (int device_index = 0; device_index < device_count; device_index++) {
    for (int channel_index = 0; channel_index < XHPTDC8_TDC_CHANNEL_COUNT; ↩
        channel_index++)
    {
      mgr_cfg->device_configs[device_index].trigger_threshold[channel_index]
        = XHPTDC8_THRESHOLD_N_NIM;
      mgr_cfg->device_configs[device_index].channel[channel_index].enable = true;
    }
    mgr_cfg->device_configs[device_index].adc_channel.enable = 1;
    mgr_cfg->device_configs[device_index].adc_channel.watchdog_readout = 0;
    mgr_cfg->device_configs[device_index].adc_channel.trigger_threshold
      = XHPTDC8_THRESHOLD_N_NIM;

    // configure an auto trigger at 150 kHz
    mgr_cfg->device_configs[device_index].auto_trigger_period = 1000;
    mgr_cfg->device_configs[device_index].auto_trigger_random_exponent = 0;

    // set all TiGers to create a short pulse for every auto trigger
    for (int block_index = 0; block_index < XHPTDC8_TDC_CHANNEL_COUNT; block_index++)
    {
      int channel_offset = block_index * 2;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].extend = false;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].negate = false;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].retrigger = ↩
          false;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].sources
        = XHPTDC8_TRIGGER_SOURCE_AUTO;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].mode = ↩
          XHPTDC8_TIGER_BIPOLAR;

      // every channel pulses a little later than the previous channel, for one clock↩
           cycle
      mgr_cfg->device_configs[device_index].tiger_block[block_index].start = ↩
          general_offset +  channel_offset;
      mgr_cfg->device_configs[device_index].tiger_block[block_index].stop = ↩
          general_offset + channel_offset + 1;

      // block trigger that is outside start-stop range
      mgr_cfg->device_configs[device_index].gating_block[block_index].negate = false;
      mgr_cfg->device_configs[device_index].gating_block[block_index].sources = ↩
```

```
          XHPTDC8_TRIGGER_SOURCE_AUTO;
      mgr_cfg->device_configs[device_index].gating_block[block_index].mode = ↩
          XHPTDC8_GATE_ON;
      mgr_cfg->device_configs[device_index].gating_block[block_index].start = ↩
          general_offset + channel_offset - epsilon;
      mgr_cfg->device_configs[device_index].gating_block[block_index].stop = ↩
          general_offset + channel_offset + epsilon + 1;
    }
  }
  return xhptdc8_configure(mgr_cfg);
}


void print_device_information() {
  xhptdc8_static_info staticinfo;
  printf("-------------------------\n");
  for (int i = 0; i < get_device_count(); i++) {
    xhptdc8_get_static_info(i, &staticinfo);
    printf("Board Serial         : %d.%d\n", staticinfo.board_serial >> 24, ↩
        staticinfo.board_serial & 0xffffff);
    printf("Board Configuration  : %d\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);
  }
}

void print_hit(TDCHit* hit) {
  bool adc_data = ((hit->channel % 10) == 8) || ((hit->channel % 10) == 9);
  if (hit->type & XHPTDC8_TDCHIT_TYPE_ERROR)
    printf("Error:\n");

  printf("Channel %u - Time %llu - Type %x", hit->channel, hit->time, hit->type);
  if (adc_data)
    printf(" - ADC Data : %d", (int)(hit->bin));

  printf("\n");
}

// call read_hits() once per millisecond until there is some data or max count of ↩
    trials
int poll_for_hits(TDCHit* hit_buffer, size_t events_per_read) {
  int trys_to_read_hits = 0;
  while (trys_to_read_hits < MAX_TRYS_TO_READ_HITS) {
    unsigned long hit_count = xhptdc8_read_hits(hit_buffer, events_per_read);
    if (hit_count)
      return hit_count;
    Sleep(1);
    trys_to_read_hits++;
  }
  if (trys_to_read_hits == MAX_TRYS_TO_READ_HITS)
```

```
    printf("not enough data, check trigger source and device configuration\n");
  return trys_to_read_hits;
}


// fetch hits from the board by polling
void read_hits_wrapper(int events_per_read) {
  int total_event_count = events_per_read * 10000;

  TDCHit* hit_buffer = new TDCHit[events_per_read];

  int total_events = 0;
  while (total_events < total_event_count) {
    unsigned long hit_count = poll_for_hits(hit_buffer, events_per_read);

    for (unsigned int i = 0; i < hit_count; i++)
    {
      TDCHit* hit = &(hit_buffer[i]);
      print_hit(hit);
      if ((total_events++ % 100) == 0)
        printf("Sum: %d - Packet events: %d\n", total_events, hit_count);
    }
  }

  delete[] hit_buffer;
}

// utility function to check for error, print error message and exit
int exit_on_fail(int status, const char* message) {
  if (status == XHPTDC8_OK)
    return status;
  printf("%s: %s\n", message, xhptdc8_get_last_error_message(0));
  exit(1);
}

int main(int argc, char* argv[]) {
  printf("cronologic xhptdc8_user_guide_example using driver: %s\n", ↩
     xhptdc8_get_driver_revision_str());
  printf("\n\nThis is illustrating the usage of an xHPTDC8 examplary with internal ↩
     triggering");

  //init manager for devices
  int error_code = initialize_xhptdc8(8 * 1024 * 1024);

  exit_on_fail(
    //configure all devices with that manager
    configure_xhptdc8(get_device_count()),
    "Could not configure.");

  print_device_information();

  exit_on_fail(
    //start measurement
    xhptdc8_start_capture(),
    "Could not start capturing.");
```

```
177
178   exit_on_fail(
179     //start TiGer-functionality
180     xhptdc8_start_tiger(0),
181     "Could not start TiGer.");
182
183   //collect measured data
184   read_hits_wrapper(10000);
185
186   exit_on_fail(
187     //stop measurement
188     xhptdc8_stop_capture(),
189     "Could not stop capturing.");
190
191   exit_on_fail(
192     //close manager
193     xhptdc8_close(),
194     "Could not close devices-manager.");
195
196   return 0;
197 }
```

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 Performance

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 $13.02083\overline{3}$		ps ps
t_{DPfull}	Interval between edges	5			ns
$f_{Readout}$	Readout rate			48	MHits/s

8.1.2 Time Base

Symbol	Parameter	Min	Typical	Max	Units
ΔT	Temperature stability 20°C to 70°C		10	25	ppb
F	Initial calibration		500	1000	ppb
$\Delta F/F_1$	Aging per year		100	200	ppb
$\Delta F/F_{10}$	Aging 10 years		1000	2000	ppb

8.1.3 ADC

Symbol	Parameter	Min	Typical	Max	Units
t_{ADC}	Interval between ADC triggers	300			ns
f_{-3dB}	-3dB bandwidth		200		MHz
V_{LSB}	Voltage resolution		76		μV

8.2 Electrical Characteristics

8.2.1 Power Supply

Symbol	Parameter	Min	Typical	Max	Units
P_{total}	Total power consumption			20	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			1500	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 xHPTDC8'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} – 1.32		V_{Base} + 1.18	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).

All digital inputs can output AC coupled pulses from the TiGer. Special care should be taken not to enable the TiGer output when sensitive equipment is connected that could be damaged by the pulses. See Section 3.3.

8.2.3 ADC Inputs

The ADC input is DC coupled to a differential termination voltage of 400 mV. This means that the actual voltage seen by the ADC will depend on the output impedance of the source that is driving the input.

Symbol	Parameter	Min	Typical	Max	Units
V_{in}	Input voltage	-2.0		2.5	V
V_{term}	Termination voltage		0.4		V
Z_{in}	Input Impedance		50		Ω

8.2.4 Clock input J2

The clock input J2 is single ended and AC coupled.

Symbol	Parameter	Min	Typical	Max	Units
V_{p-p}	Peak to Peak voltage	1		3.3	V
V_{cm}	Common mode voltage voltage	-3		3	V
V_{tck}	Clock termination voltage		0		V
Z_{in}	Input Impedance		50		Ω
D_{J2}	Duty cycle	45	50	55	%
f_{J2}	Frequency		10		MHz

8.3 Information Required by DIN EN 61010-1

8.3.1 Manufacturer

The xHPTDC8 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 xHPTDC8, 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 xHPTDC8 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 with regards to 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 3.3). Doing so could do permanent damage to the xHPTDC8 and the external hardware.

8.3.5 Cooling

The xHPTDC8 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 xHPTDC8 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 xHPTDC8 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 45.

9 Revision History

User Guide 1.8.14 as of 2024-03-27
cronologic GmbH & Co. KG
Jahnstraße 49
60318 Frankfurt am Main
Germany

9.1 Firmware

Revision	Date	Comments
2.1148	2022-04-21	Multi card mode improvements
2.1140	2022-04-07	Fixed timestamp offset issues when using multiple devices
2.1134	2021-12-10	Fixed TDC overtemp alarm issue
2.1124	2021-10-13	Fixed TDC alignment issues
2.1114	2021-06-11	Fixed ADC packet ordering
2.1110	2021-05-25	Fixed TDC packet building
2.1108	2021-05-07	Fixed rare lock-up during PCIe configuration reads
2.1089	2021-03-31	Fixed timestamp offsets between channels
2.1054	2021-02-04	Preliminary prototyping firmware

9.2 Driver & Applications

Revision	Date	Comments
1.1.0	2023-12-19	Internal code improvements Added new structure `crono_pcie_info`
1.0.9	2023-11-07	Fixed a bug on reading cronorom
1.0.8	2023-09-07	Prevent overwriting of the FPGA bitstream writing the user flash Multiple internal improvements
1.0.3	2022-09-07	Various Improvements cronologic kernel mode driver 1.4.2
1.0.2	2022-08-29	Various bug fixes cronologic kernel mode driver 1.4.1
1.0.1	2021-12-08	Primary release

9.3 User Guide

Revision	Date	Comments
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 Userguide 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
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