# Time-Over-Threshold Logic for the AGATA Digitizer

Ian Brawn, 1 July 2010.

## 1 Overview

The logic described in this document has been developed to add functionality to the Core firmware of the AGATA Digitizer. It enables the Digitizer to extract Time-Over-Threshold (TOT) data from the pre-amplifiers by measuring the duration of the *inhibit* signal. The extracted data are then transmitted to the Pre-Processor via the existing RocketIO links (which normally carry ADC data).

As there is no spare capacity on the RocketIO links it is necessary to suppress the transmission of some ADC data to allow the transmission of TOT data. Full details of the protocol used are given in section 2.

The TOT logic comprises four logic blocks: the TDC, TOT Transmitter, TOT Receiver and Slow-Control Interface. As shown in figure 1, all four of these blocks are instantiated in the Digitizer firmware, and the TOT Receiver block should also be instantiated in the Pre-Processor firmware. The functionality of these blocks is as follows.

- The TDC measures the duration for which the *inhibit* signal is asserted.
- The TOT Transmitter receives data from the TDC; it attaches a header to the data and sends the resultant packet to the RocketIO transceiver for transmission across the link to the Pre-Processors.
- The TOT Receiver receives data from the RocketIO transceiver. It extracts TOT data from the data stream, presenting them on a dedicated output and masking them from the downstream logic on the ADC-data path. The instance of the TOT Receiver in the Digitizer firmware is for diagnostic purposes.
- The TOT Slow-Control Interface provides a slow-control interface to all other blocks of TOT logic instantiated in the Digitizer firmware.

Note that, whereas there is one TDC per Digitizer, the TOT Transmitters and Receivers are instantiated per channel (with the same TOT information being sent to both channels). Furthermore, the Slow-Control Interface is composed of two blocks of identical logic, each providing the interface for the TOT logic in one channel. Sections 3–4 of this document describe all of these logic blocks in more detail.

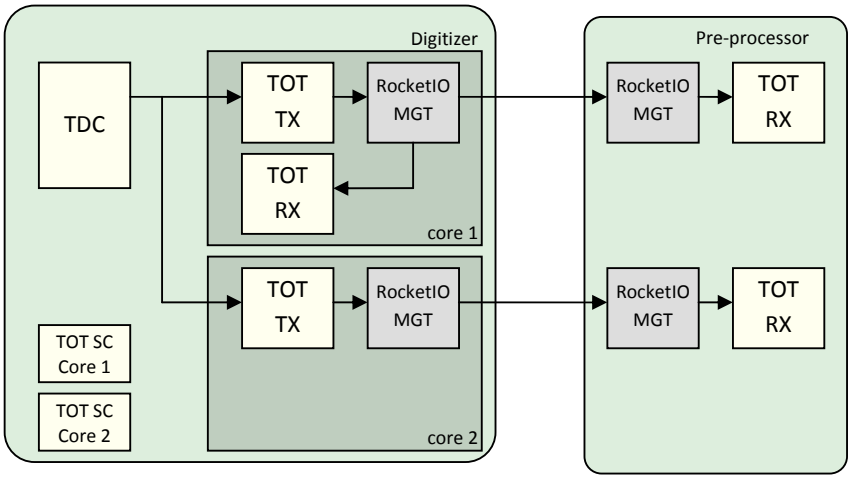


Figure 1. The organisation and modularity of the four blocks of TOT logic: the TDC, the TOT Transmitter (TX), the TOT Receiver (RX) and the Slow-Control Interface (TOT SC).

# 2 TOT Transmission Protocol

The protocol for transferring Time-Over-Threshold (TOT) data from the Digitizer to the Pre-Processor is as follows.

As the full capacity of the Digitizer–Pre-processor links is used transmitting ADC, *sync* and *inhibit* data, it is necessary to suppress some of this data to allow the transmission of TOT data. This is done during the period of dead time that follows an active *inhibit* signal, the cause and character of which is given below.

The dead time exists because the ADC values captured whilst *inhibit* is active are non-physical. Downstream in the Pre-processor, in the Moving Window De-convolution (MWD) algorithm, there is a pipeline memory that holds a history of ADC values. Only when all non-physical ADC values have passed from this pipeline does this algorithm resume producing valid results. There is thus a period of dead time following an active *inhibit* signal during which no valid pulse information is extracted from the ADC data. During this period ADC data can be suppressed and TOT data transmitted with minimal consequences. It is desirable, however, that the transmission protocol doesn't result in further non-physical data being sent to the MWD algorithm, in order that the dead time not be lengthened.

The dead time is known to be at least one microsecond in duration, so it is a specification of the TOT transmission that it should be completed within this period. (In practice the transmission should normally be completed within 150 ns of the trailing edge of *inhibit*).

A TOT measurement is four bytes in size. Using the full data width available on the Rocket IO link this requires two (RX/TX) clock cycles to transfer. The TOT measurement is therefore transferred on two consecutive clock cycles, with the following bit order:

- TOT word 1: the first TOT word transmitted comprises the 16 least-significant bits of the full, 32-bit TOT word. Bit 0 is the least significant.
- TOT word 2: the second TOT word transmitted comprises the 16 most significant bits of the 32-bit TOT word. Bit 0 is the least significant (i.e., bit 0 of TOT word 2 equates to bit 16 of the full, 32-bit TOT word).

The meaning of the bits within the full 32-bit TOT word is specific to the TDC implementation and is described in section 5.

The TOT words are distinguished from normal data by a preceding K character, *K28.0*, a control character from the 8B/10B encoding set used by the RocketIO link. The transfer of each TOT measurement thus has the form shown in Figure 2. As the RocketIO data bus is two bytes wide and *K28.0* is only one byte wide, it is transmitted on both the upper and lower bytes of the data bus simultaneously.

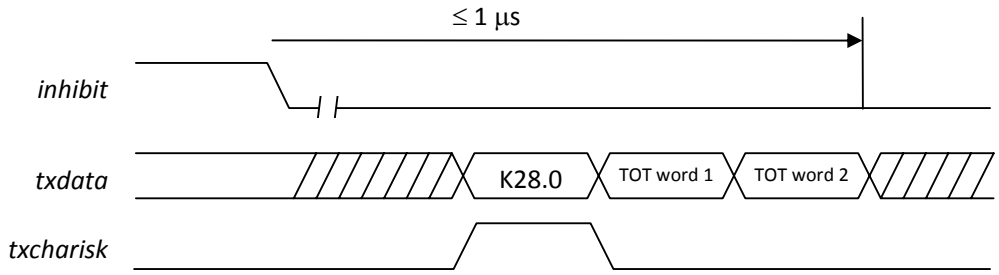


Figure 2. The format of the TOT packet as seen at the input to the RocketIO transceiver on the Digitizer. Here, *inhibit* is the signal from the pre-amplifier, *txdata* is the 16-bit data input to the transceiver, and *txcharisk* is the control signal to the transceiver that flags *K* characters.

At the Pre-Processor, the signal *rxcharisk* from the RocketIO receiver, together with the decoded *K* character on *rxdata*, is used to trigger both the capture of the subsequent two words of TOT data and the masking of the TOT packet from the downstream data path (including the MWD algorithm).

# 3 The TOT Transmitter Logic

The TOT transmitter logic, which is shown in figure 3, sits on the real-time data path in the Digitizer firmware, directly upstream of the RocketIO MGT transceiver block. The functionality is described below.

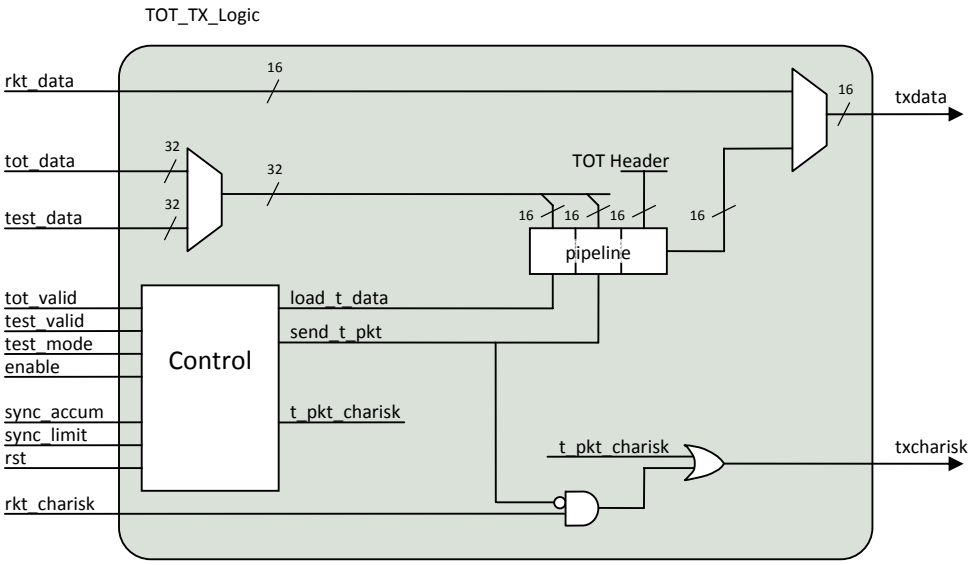


Figure 3. A functional representation of the TOT Transmitter logic.

If the TOT Transmitter Logic is not enabled (i.e., *enable* is low), it passes the data on *rkt_data* and *rkt_charisk* transparently to the outputs *txdata* and *txcharisk* respectively (with no added latency).

If the TOT Transmitter Logic is enabled and not in test mode (i.e., *enable* is high and *test_mode* is low), it responds to the assertion of *tot_valid* by generating a TOT packet of the format shown in figure 2. The data sent in the TOT packet are those presented on *tot_data* synchronously with the assertion of *tot_valid*. As described in section 2, the least-significant byte of this word is transmitted first.
The TOT packet is transmitted on *txdata* and *txcharisk*, replacing the ADC data from *rkt_data* and *rkt_charisk* that would otherwise be sent. There is minimum latency of one clock cycle from the assertion of *tot_valid* to the appearance of the TOT header on *txdata.* However, the latency for a particular instance may be longer as the transmission of TOT packets is delayed if any of the following conditions are true:

1. a sync signal is due in the next four clock cycles (as determined by the inputs *sync_accum* and *sync_limit*);

2. *rkt_data*(15) (*sync/inhibit*) is currently asserted, or has been asserted in any of the three previous clock cycles;

3. either of the *rkt_charisk* bits is asserted, or has been asserted in any of the three previous clock cycles;

4. fewer than four clock cycles have elapsed since the TOT Transmitter output the last word of the previous TOT packet.

Only when none of these conditions are true is a TOT packet sent. The delay thus added to some TOT packets serves two purposes: firstly, it ensures that a *sync* signal is never replaced by a TOT packet; secondly, it ensure that *sync* signals, *K* characters and TOT data are not repeated in the data stream by the TOT Receiver logic (see section 4).

The TOT Transmitter asserts *busy* on receipt of an asserted *tot_valid* signal and only clears it when the output of the TOT packet on to *txdata* and *txcharisk* has been completed. Whilst *busy* is asserted the TOT Transmitter will not respond to any further assertions of *tot_valid* (nor to any changes of *tot_data*, as the data to be transmitted are latched on receipt of the initial *tot_valid* assertion).

If the TOT Transmitter Logic is enabled and in test mode (i.e., *enable* and *test_mode* are both high), it responds to the inputs *test_valid* and *test_data* in exactly the same fashion as it responds to *tot_valid* and *tot_data* in normal mode. That is, the data presented on *test_data* are output onto *txdata* in a TOT packet with all the attendant functionality described above.

In test mode the TOT Transmitter will not respond to assertions of *tot_valid* and, conversely, when not in test mode it will not respond to assertions of *test_valid*.

# 4 The TOT Receiver Logic

Figure 4 shows a functional representation of the logic required to receive the TOT information in the Pre-Processor. It is intended that this logic should sit directly on the output of the RocketIO MGT transceiver block and process the *rxdata* and *rxcharisk* signals before they are received by any other logic in the Pre-Processor.

By default the TOT Receiver passes the data on inputs *rxdata* and *rxcharisk* to the corresponding output ports transparently (with a fixed latency of one clock cycle). However, if the logic is enabled and a TOT header (*K28.0*) is received, the logic performs the following tasks:

- latches the two 16-bit words following the *K* character onto the 32-bit bus *TOT_data*;
- asserts the signal *TOT_flag* for one clock cycle when there is new, valid data on *TOT_data*;
- masks the TOT header and the following two data words from the output bus *data_out* by sending copies of the three consecutive data words that preceded the *K* character;
- masks the active *rxcharisk* flag accompanying *K28.0* from the signal *charisk_out*.

When the TOT Receiver logic is not enabled, all *rxdata* and *rxcharisk* data are passed through the logic transparently, including any *K* characters and TOT data received.

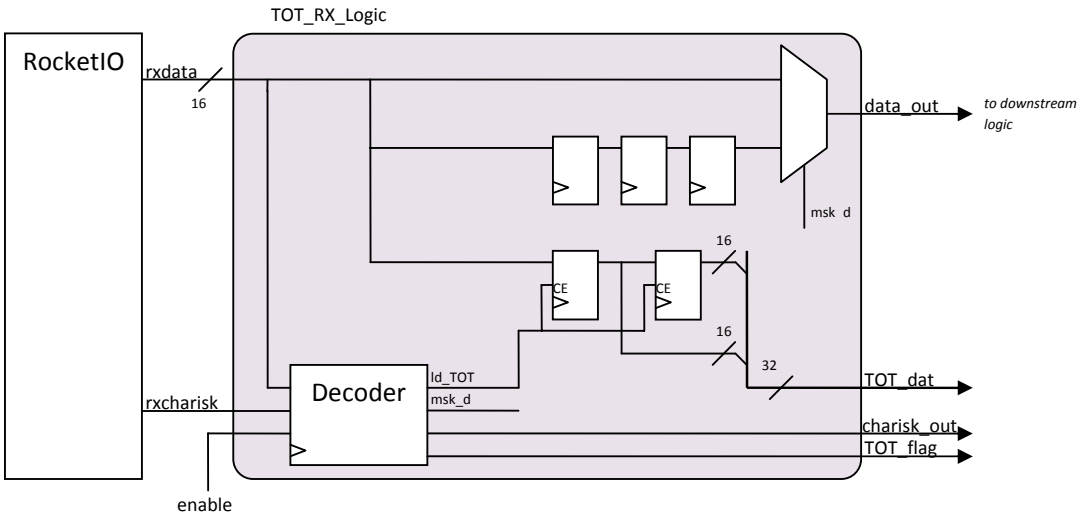


Figure 4. The block TOT_RX_Logic shows a functional representation of the logic required in the Pre-Processor to extract TOT data from the data stream received from the RocketIO transceiver.

The resultant data on *rxdata, data_out, TOT_data* and *TOT_flag* are illustrated in Figure 5. Note the latency from *rxdata* to *data_out* is one clock cycle.

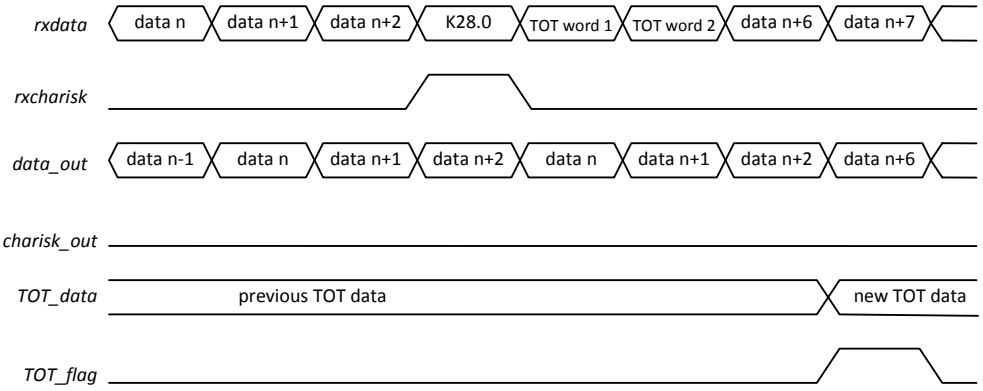


Figure 5. The real-time input and output signals of the TOT receiver logic in the Pre-processor.

Any *K* characters other than the nominated TOT header (*K28.0*) are passed transparently though the TOT Receiver logic, together with the attendant *rxcharisk* flag. At present only one other *K* character is used by the Digitizer–Pre-Processor link: *K28.7* is used for data alignment.

Note that the TOT receiver logic does nothing to prevent *sync* signals, *K* characters or previous TOT data being captured and repeated by the TOT masking process. That is, it does nothing to prevent these data occupying the positions *data n* to *data n*+2 in Figure 5. This is not necessary as the TOT transmission logic in the Digitizer co-ordinates the transmission of TOT packets and these special data to ensure that such situations do not arise (see section 3).

In addition to being required in the Pre-Processor the TOT Receiver is instantiated in the loopback RocketIO path of the Digitizer, for test and diagnostic purposes.

# 5 The TOT TDC

The implementation of the TDC is based on the interpolation method of time extraction as described in the document *TOT measurement*, by Laurent Charles. The output of the TDC is the 32-bit word, *tot_data*, which comprises the following three elements:

bits 0–15: $T_{course}$: the number of whole clock cycles for which *inhibit* was asserted;
bits 16–23: $T_{fine}$: the sub-clock-period adjustment ;
bits 24–31: $T_{ref}$: a reference measurement of 1 clock cycle to calibrate the above.

Here the sub-clock measurement, $T_{fine}$, is a signed number, with negative numbers given in two's complement. All of the other fields are unsigned, positive numbers.

The duration of the *inhibit* signal can be calculated from these measurements thus:

*inhibit* duration = $(T_{course} + T_{fine} / T_{ref}) \times 5$ ns

where 5 ns is the period of the TDC clock.

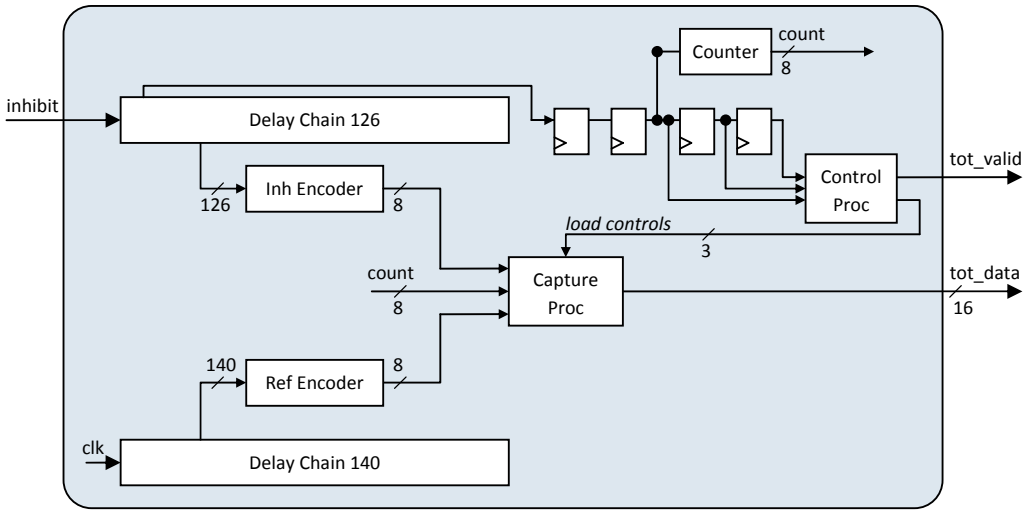


Figure 6. A block diagram of the TOT TDC.

Figure 6 shows a block diagram of the TOT TDC. The blocks can be grouped into four main areas of functionality:

- the measurement of Tcourse (Counter),
- the determination of $T_{fine}$ (Delay Chain 126, Inh Encoder, Capture Proc),
- the measurement of $T_{ref}$ ((Delay Chain 140, Ref Encoder, Capture Proc),
- the generation of control signals (Control Proc).

Each of these functions and the attendant areas of logic are described in the sections below. These sections are followed by one describing the implementation of the delay chains.

## 5.1 The measurement of $T_{course}$

This measurement is achieved using a simple counter, clocked at 200 MHz.

As indicated in figure 6, the version of the *inhibit* signal measured by the counter is not sourced directly from the input port, but rather from the first (clocked) tap of Delay Chain 126. This is done to minimize the fan-out and capacitive load of the incoming *inhibit* signal, with the aim of producing a more accurate measurement of $T_{fine}$ (see below). This clocked version of the *inhibit* signal is also passed through a number of further flip-flops on route to the counter, in order to minimize problems of metastability.

## 5.2 The determination of $T_{fine}$

The sub-clock adjustment, $T_{fine}$, is calculated from two individual measurements:

$T_{fine} = T_{rising} - T_{falling}$

where $T_{rsising}$ is the duration between the rising edge of *inhibit* and the subsequent TDC clock edge, and $T_{falling}$ is the duration between the falling edge of *inhibit* and the subsequent clock edge. These intervals are illustrated in figure 7.

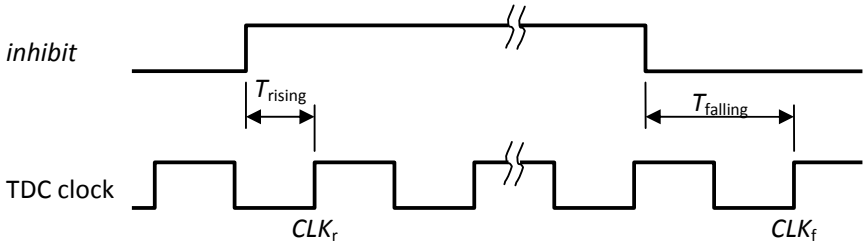


Figure 7. An illustration of the intervals $T_{rising}$ and $T_{falling}$.

To measure $T_{rising}$ and $T_{falling}$ the incoming *inhibit* signal is fed into a chain of 126 asynchronous delay elements (Delay Chain 126 in figure 6). This delay chain is tapped after each element and the taps are clocked and sent to a binary encoder (see figure 8). The output of the encoder is an 8-bit count of the number of taps asserted, which gives a measure of the duration before that clock edge for which *inhibit* was asserted. When the encoder is read on clock edge $CLK_r$ in diagram 7, this measurement corresponds to $T_{rising}$. When the encoder is read on clock edge $CLK_f$, the measurement corresponds to the complement of $T_{falling}$. That is, $T_{falling}$ can be calculated from the maximum possible number of taps asserted (126) minus the measured number of taps asserted.

The process Capture Proc, shown in figure 6, registers the encoder values at the appropriate times and performs the calculations necessary to obtain $T_{rising}$, $T_{falling}$ and hence $T_{fine}$. It is driven by signals from Control Proc (see below).

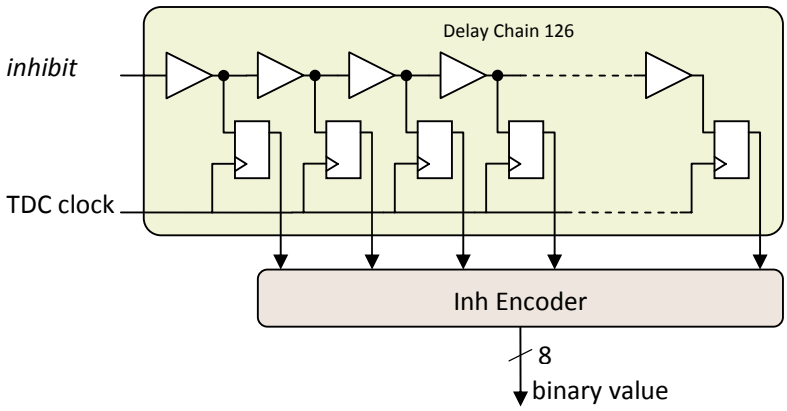


Figure 8. The *inhibit* delay chain and encoder.

## 5.3 The measurement of $T_{ref}$

To calibrate the measurement $T_{fine}$ it is necessary to compare it to the measurement of a known period. This reference measurement is obtained by feeding the TDC clock into a second delay chain, Delay Chain 140, which runs parallel to the *inhibit* delay chain in the FPGA fabric. As with the *inhibit* chain, the reference chain is tapped and these signals are clocked and sent to an encoder. However, here the encoder does not perform a simple count of the number of asserted taps, but performs the more complex calculation described below.

The reference encoder does not examine the full range of taps presented by Delay Chain 140. It examines only two windows that are centred on taps where consecutive rising edges of the TDC clock are expected to be recorded. As the test signal in this case is the same as the sampling clock, these rising edges will always be recorded at approximately the same place in the chain. The position and width of the windows used by the reference encoder are fixed, but have been determined by experimentation to give the least sensitivity to clock jitter and drift.

Within these two windows the reference encoder counts the number of asserted taps, to obtain a measure of the two intervals $T_1$ and $T_2$ shown in figure 9. From these, and the fixed offset between the two sampling windows, $T_{woff}$, the clock period $T_{ref}$ can be calculated:

$T_{ref} = T_{woff} + T_1 - T_2$

This measurement is captured by Capture Proc at the end of an *inhibit* signal, synchronously with the capture of the $T_{falling}$ measurement.

Note that in the document *TOT measurement*, Laurent Charles proposes measuring $T_{ref}$ by measuring the duration for which the clock signal is asserted and doubling this figure. Such a method is only suitable if the duty cycle of the clock is at or close to 50%. Generally, this will not be the case if the clock has been generated from one of a different frequency using a DCM. For the TDC logic described here the clock is indeed generated in this way and it has a duty cycle of approximately 30%. For this reason $T_{ref}$ must be measured by examining the full clock cycle rather than just the active part of it.

## 5.4 The generation of control signals

All of the control signals in the TDC logic are generated from the *inhibit* signal. For this purpose the *inhibit* signal is sourced from the first tap of Delay Chain 126 and fed into a pipeline of flip-flops, which hold a history of the signal. The process Control Proc examines this history and generates the necessary control signals in relation to the rising and falling edge of *inhibit*. From the rising edge it generates a load signal to capture $T_{rising}$; from the falling edge it generates signals to capture $T_{falling}$ , $T_{ref}$ and $T_{course}$, and it also generates *tot_valid*. These signals are timed to account for the different latencies of the various measurements.

To prevent TOT packets being generated for *sync* signals, which are transmitted on the same line as *inhibit* but are only 10 ns long, *tot_valid* is generated only if *sync*/*inhibit* is asserted for at least 20 ns (4 ticks of the TDC clock).

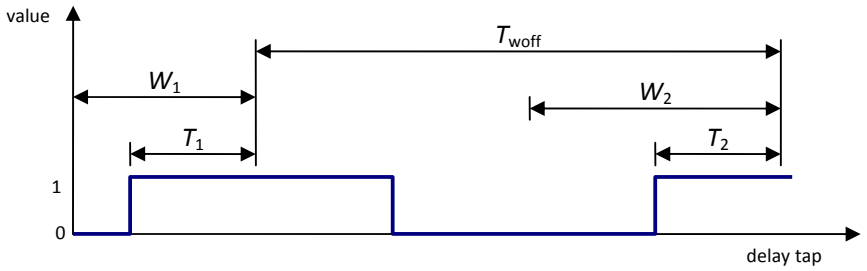


Figure 9. The waveform of the TDC clock as seen in the reference delay chain. Shown are the two windows of taps examined, $W_1$ and $W_2$, the two measurements obtains from these windows, $T_1$ and $T_2$, and the offset between the two windows, $T_{woff}$.

## 5.5 The implementation of the delay chains

The two delay chains in the TDC, Delay Chain 126 and Delay Chain 140, are built using Xilinx fast-carry chains. In order to obtain the necessary degree of consistency in the propagation delays down the chains, they are implemented as relationally-placed macros. The two chains are placed contiguously in the FPGA fabric, running parallel to each other. For more detail on their structure see the document *TOT measurement* by Laurent Charles, which was used as a guide in their construction.

To obtain valid measurements from the chains it is necessary that the propagation delay down the entire chain be longer than one period of the TDC clock. In hardware tests the 5 ns clock period of the TDC clock was measured as 105 ± 0.4 delay elements when the device was cold (and 104 ± 0.8 elements when the device was hot). For the *inhibit* delay chain (Delay Chain 126) this gives an overhead of about 20 delay elements, or 16%. The reference delay chain is implemented using a slightly longer chain than this, of 140 elements. This is necessary for the optimum placement of the measurement windows described in section 5.3.

# 6 The Digitizer TOT Control Interface

The following is the slow control interface to the TOT-logic in the Digitizer firmware. Most of the control and status bits here have been implemented in new registers occupying previously vacant locations in the address map, as specified in EDOC705, Version date 25th June 2009. In one case a new bit has been added to an existing Core register (bit 15 of the Core Control Register, which was previously unused). For this register the bits that existed previously, and are unrelated to the TOT logic, are shown in grey.

Note that some bits and registers are only implemented for channels where the loopback and TOT Receiver logic are present (namely, Core channel 1). For channels without TOT Receiver logic those register bits are unused; writing to them will have no effect and they will always read zero. Any bits not explicitly specified in a register are also unused and will respond in the same fashion.

**Control Register – *0x0000 0000***

0. Laser Enable : set to enable laser
1. Laser Disable : set to disable laser
2. Laser Resetn : clear to reset Laser
3. Operates LED 7.
4. shdn_c : controls pre-amp signal
5. Laser RX enable : set to enable the receiver part of the Laser
6. Laser SQ enable : set to enable the ?????? part of the laser .
7.
8. Reset other DCMs to allow lock after connect the optical link
9. Set to 1 to enable the inhibit signal to be transmitted.
10.
11. Set to 1 to reset the Trigger Block logic.
12. Set to 1 to reset the Sprom and re-read all the contents into RAM
13. Set to 1 to enable the RAM test.
14.
15. Set to 1 to reset the TOT TDC logic

***TOT TX/RX channel 1 Registers – 0x0000 0050 to 0x0000 0056***

**Status – Offset 0x0**

***Read Only***

0. TOT Rx Busy

**Control Mode – Offset 0x1**

***Read/Write***

0: Enable TOT Tx
1: Enable TOT Rx (where loopback implemented for channel)
2: TOT Tx test mode

**Control Pulse – Offset 0x2**

***Always zero on read***

0: *test_valid* (initiates transmission of test data from TOT Tx).

4: Reset TOT Tx (and Rx if implemented for this channel).

**Test Data LSB – Offset 0x3**

***Read/Write***

0–15: Lower 2 bytes of test data to transmit from TOT Tx.

**Test Data MSB – Offset 0x4**

***Read/Write***

0–15: Upper 2 bytes of test data to transmit from TOT Tx.

**Data Rx LSB – Offset 0x5**

***Only implemented for channel with loopback logic***

***Read Only***

0–15: Lower 2 bytes of most recent data received at TOT Rx.

**Data Rx LSB – Offset 0x6**

***Only implemented for channel with loopback logic***

***Read Only***

0–15: upper 2 bytes of most recent data received at TOT Rx.

***TOT TX/RX channel 2 – 0x0000 0060 to 0x0000 0066***

***Details as per TOT TX/RX channel 1.***