White Rabbit Electrical Absolute Calibration Procedure

P.P.M. Jansweijer (peterj@nikhef.nl)
H.Z. Peek (henkp@nikhef.nl)
T.J. Pinkert (t.j.pinkert@alumnus.utwente.nl)
G.C Visser (guidov@nikhef.nl)

Version 1.1
August 28, 2018

The work described in this document is part of the ASTERICS infrastructure cluster which is a part of the European Horizon 2020 program http://asterics2020.eu/
European Commission grant no 653477
## Contents

1. Introduction ........................................................................................................................ 4
   1.1. General ......................................................................................................................... 4
   1.2. Overview ..................................................................................................................... 4
   1.3. Definition ..................................................................................................................... 5
2. Electrical absolute calibration principle ............................................................................. 6
   2.1. Overview ..................................................................................................................... 6
   2.2. How to measure $t_A$ ................................................................................................... 8
   2.3. TIC measurement ........................................................................................................ 8
   2.4. Oscilloscope measurement .......................................................................................... 9
   2.5. Combining the measurement results .......................................................................... 11
3. Equipment requirements ..................................................................................................... 12
   3.1. Check absolute calibration capability ........................................................................ 12
   3.2. Equipment list ............................................................................................................ 12
   3.3. Software ..................................................................................................................... 12
4. Electrical Absolute Calibration Procedure ....................................................................... 13
   4.1. 10MHz Reference Clock ........................................................................................... 13
   4.2. Connect the TIC and DSO to Ethernet ...................................................................... 13
   4.3. Loopback Tx to Rx electrical interface ..................................................................... 14
   4.4. Measure the skew of your measurement setup .......................................................... 14
      4.4.1. TIC Skew ............................................................................................................ 15
      4.4.2. DSO Skew .......................................................................................................... 17
   4.5. Measuring $t_A$ ........................................................................................................... 18
      4.5.1. TIC measurement: PPS to abscal_txts ............................................................... 19
      4.5.2. DSO measurement: abscal_txts to Tx packet SFD ............................................ 22
   4.6. Bringing it all together ............................................................................................... 24
   4.7. Apply the calibration constants ................................................................................. 25
5. Appendix A .................................................................................................................... 26
   Guide to use other oscilloscopes .......................................................................................... 26
   ```python
   get_waveforms(scope, channels=[1,2,3,4],num_avg=1,output_dir="data")
   ```
   ```python
   file_to_waveform(filename)
   ```
   ```python
   preamble_string_to_dict(preamble_string)
   ```
   ```python
   raw_to_scipy_array (waveform_raw, byte_order, preamble)
   ```
osc_init(scope, time_base = 50.0e-9) ................................................................................... 28
check_waveforms(waveform_data) ..................................................................................... 28

6. Appendix B ...................................................................................................................... 29
   Calibration of an FMC module. ........................................................................................ 29

7. Appendix C ...................................................................................................................... 30
   PHY word width ............................................................................................................. 30

8. References ...................................................................................................................... 31

Table 1: List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSO</td>
<td>Digital Sampling Oscilloscope</td>
</tr>
<tr>
<td>EO</td>
<td>Electrical Optical converter</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GPSDO</td>
<td>GPS Disciplined Oscillator</td>
</tr>
<tr>
<td>MAC</td>
<td>Media Access Control</td>
</tr>
<tr>
<td>MTP</td>
<td>Message Timestamp Point</td>
</tr>
<tr>
<td>OE</td>
<td>Optical Electrical converter</td>
</tr>
<tr>
<td>PPS</td>
<td>Pulse Per Second</td>
</tr>
<tr>
<td>PTP</td>
<td>Precision Time Protocol</td>
</tr>
<tr>
<td>SFD</td>
<td>Start of Frame Delimiter</td>
</tr>
<tr>
<td>SFP</td>
<td>Small Form factor Pluggable</td>
</tr>
<tr>
<td>TDR</td>
<td>Time Domain Reflectometry</td>
</tr>
<tr>
<td>TIC</td>
<td>Time Interval Counter</td>
</tr>
<tr>
<td>WR</td>
<td>White Rabbit</td>
</tr>
</tbody>
</table>
1. Introduction

1.1. General

This document describes the electrical absolute calibration procedure for White Rabbit (WR) devices. Chapter 1 gives an overview of absolute calibration while chapter 2 focusses on the principles of electrical absolute calibration. A list of equipment and pointers to software is provided in chapter 3. Proceed immediately to chapter 4 if you are only interested in a step by step guide to measure the electrical absolute calibration parameters for your specific WR device.

1.2. Overview

According to the Precision Time Protocol [1], WR devices timestamp the transmission and reception of messages in hardware. Delays exist between the timestamp point in the interior of the WR device and its time reference planes, i.e. the connectors of the device. For high accuracy time transfer these delays must be calibrated and taken into account.

The current relative WR calibration procedure [2] calibrates WR devices plus electrical-optical/optical-electrical converter as a single entity. One entity is promoted to be a “Golden Standard”. In practice it is difficult to keep such a standard stable and alive. Worldwide derivatives have to be distributed for calibration. These issues are dealt with when devices are absolute calibrated.

Absolute calibration is based on (SI: International System of Units) time measurements and is not based on a “Golden Standard”. Instead of calibrating the system as a whole, the individual network components can be separately calibrated. A calibrated system can be created by using the individually calibrated components. This also enables plug&play exchange of these components without loosing calibration. Onsite network calibration can be avoided when using absolute calibrated components. Absolute calibration enforces standardization since it enables independent developers and/or vendors to exchange their WR gear while achieving absolute sub ns timing.

Absolute calibrated components can function as “Golden Standards” for the relative calibration procedure [2] that usually is easier to perform. As such relative- and absolute-calibration procedures are both valuable and should co-exist.

Accuracy decreases with each calibrator generation. The device calibration source should be traceable by a unique primary calibration device identification number (for instance, its MAC address) and a calibration generation number.

Electrical absolute calibration is achieved by measuring the time relationship ($\Delta_{TXcal}$, $\Delta_{RXcal}$) between the internal timestamps ($t_1$ and $t_{4p}$) and the external electrical time reference planes which consist of the PPS signal and the electrical interface to the electrical-optical/optical-electrical converter (EO/OE, usually the electrical SFP connector) as shown in Figure 1. EO/OE converters have their own calibration parameters that define the relationship between their electrical and optical time reference planes. This enables the exchange of EO/OE
converters without the need for re-calibration. Note: EO/OE converter calibration is outside
the scope of this document, more information can be found in reference [3].

For some WR applications the electrical system can be broken into different parts (for
example, an FMC card to be plugged onto a SPEC [4]). When the FMC module is part of the
WR device timing, then such a module needs separate calibration. The calibration parameters
must be stored in internal FMC memory, such that they are accessible by the network
management layer. FMC calibration is out of the scope of this document however Appendix B
describes how this could be done for an FMC card.

Different types of calibration exist (include or exclude FMC and/or SFP). A calibration
descriptor, stored in WR device memory, should guide the network management layer to
properly setup calibration parameters. Note that a calibration descriptor is not yet defined in
current implementations.

In this document a SPEC and DIO [5] are calibrated as an indivisible entity.

1.3. Definition

By definition an electrical absolute calibrated WR device is a combination of hardware,
FPGA firmware, embedded software and calibration constants that determine the time
relationship between the internal timestamps and external electrical time reference planes. A
new calibration is required when one of the items in the above mentioned combination is
changed, for example after FPGA firmware recompilation.
2. **Electrical absolute calibration principle**

2.1. **Overview**

Electrical absolute calibration is achieved by measuring the time relationship \((\Delta_{TXcal}, \Delta_{RXcal})\) between the internal timestamps and external electrical time reference planes which consist of the PPS signal and the electrical interface of the SFP connector. To perform electrical absolute calibration, the WR device must be in calibration mode \((mode\ abscal)\). A SFP+ timing calibration module (see picture in Figure 9), specifically designed for electrical absolute calibration of WR devices, electrically creates a loopback with a calibrated delay \((t_{loop})\) between the transmit output stream and the receive input stream of the electrical SFP connector as shown in Figure 2. The module also provides stable and repeatable access to the electrical time reference plane of the SFP connector. The serial data stream that crosses this time reference plane is probed without seriously affecting the loopback signal integrity. The probed signal is fed to a differential limiting amplifier and forwarded to two connectors. The SFP+ timing calibration module loopback- and probe-delays are stored inside the module.

![WR Abscal](image)

**Figure 2: WR device in mode abscal with loopback**

Precision Time Protocol (PTP) [1] packets are timestamped inside WR devices and hence are defined on the “internal time” scale depicted by the red arrow in Figure 3. Delays exist between the internal time scale and the external electrical time reference planes that are defined on the “external time” scale (blue arrow in Figure 3).
The external time scale uses the PPS output signal as a time reference point \((t=0)\). Define \(t_A\) as the time between the PPS signal and the Tx Message Timestamp Point (MTP), as defined in IEEE-1588 (clause 7.3.4.1 [1]), that traverses over the external electrical time reference plane. The time it takes for a \(t_1\) timestamped Tx packet to traverse the external electrical reference plane is

\[
\Delta_{TXcal} = t_A - t_1 = \Delta_{tx} - \Delta_{PPS}
\]  

When applying a calibrated loopback (i.e. SFP+ timing calibration module) from Tx to Rx then

\[
\Delta_{RXcal} = t_A + t_{loop} - t_{4p} = -\Delta_{rx} - \Delta_{PPS}
\]  

defines the time it takes for a \(t_{4p}\) timestamped Rx packet to traverse the external electrical reference plane. Note that \(t_{4p}\) includes fine-time delay phase information (see [6] for more information).

By definition, the moment a Tx packet traverses the external time reference plane is later than \(t_1\) on the internal time scale, hence \(\Delta_{TXcal}\) is positive proportional to \(\Delta_{TX}\). The opposite is true for the moment a Rx packet traverses the external time reference plane since this is by definition before \(t_{4p}\) on the internal timescale, hence \(\Delta_{RXcal}\) is negative proportional to \(\Delta_{RX}\).

The internal timescale is ahead of the external timescale by \(\Delta_{PPS}\), which is taken into account in both \(\Delta_{TXcal}\) and \(\Delta_{RXcal}\). Applying these fixed calibration parameters virtually shifts the internal time scale such that it lines up with the external time scale as is shown by the green arrowed time scale in Figure 3.

The delays \(\Delta_{TXcal}, \Delta_{RXcal}\) and \(\Delta_{PPS}\) are internal to the WR device and cannot be measured directly. However, \(t_A\) can be measured electrically and the internal timestamps \(t_1\) and \(t_{4p}\) are recorded by the WR embedded software. With \(t_A, t_1, t_{4p}\) and \(t_{loop}\) known, the calibration parameters \(\Delta_{TXcal}, \Delta_{RXcal}\) can be calculated by using equations (1) and (2).
As can be seen in Figure 3, $t_1$ on the green timescale aligns with the tx external time reference plane when $\Delta_{TXcal}$ is added while $t_{4p}$ on the green timescale aligns with the rx external time reference plane when $\Delta_{RXcal}$ is subtracted. Note that subtracting equation (2) from (1) yields:

$$\Delta_{TXcal} - \Delta_{RXcal} = t_{4p} - t_1 - t_{loop} = \Delta_{tx} + \Delta_{rx}. \quad (3)$$

### 2.2. How to measure $t_A$

To measure $t_A$, PTP Sync packets are send under software control and therefore have a non-deterministic offset with respect to the external time scale reference point, i.e. the PPS signal. This fact complicates the measurements since it means that time $t_A$ varies with each PTP message exchange.

Most (even modern) measurement equipment is not able to measure $t_A$ directly since the message time stamp point is 8B/10B encoded in a 1000BASE-X Ethernet stream. An intermediate helper signal (abscal_txts) that is derived from the internal Tx timestamp signal of the WR PTP-core [7] is used to split the measurement into:

1) a measurement to log the varying time between the PPS signal and the abscal_txts signal that is output by the WR PTP-core using a high resolution Time Interval Counter (TIC),

2) a measurement of the fixed time between abscal_txts and the 8B/10B encoded pattern for the IEEE-1588 Message Timestamp Point (MTP) using a Digital Signal Oscilloscope (DSO) that is connected to the probe output of the SFP+ timing calibration module.

The above mentioned measurements add up to $t_A$

$$t_A = TIC_{PPS\rightarrow abscal\_txts} + DSO_{abscal\_txts\rightarrow MTP}. \quad (4)$$

so equations (1) and (2) can be re-written into

$$\Delta_{TXcal} = TIC_{PPS\rightarrow abscal\_txts} - t_1 + DSO_{abscal\_txts\rightarrow MTP}, \quad (5)$$

$$\Delta_{RXcal} = TIC_{PPS\rightarrow abscal\_txts} - t_{4p} + DSO_{abscal\_txts\rightarrow MTP} + t_{loop}. \quad (6)$$

### 2.3. TIC measurement

For each TIC measurement there is a corresponding WR software $t_1$ and $t_{4p}$ timestamp. The differences between the TIC measurement and timestamps are constant (defined as $TIC_{t_1}$ and $TIC_{t_{4p}}$ respectively)

$$TIC_{PPS\rightarrow abscal\_txts} - t_1 = constant = TIC_{t_1}, \quad (7)$$

$$TIC_{PPS\rightarrow abscal\_txts} - t_{4p} = constant = TIC_{t_{4p}}. \quad (8)$$
The TIC is started by the PPS signal and stopped by helper timestamp signal abscal_txts. Figure 4 shows an example of both the TIC value and the WR software timestamps that are plotted for 180 measurements\(^1\). The absolute values vary due to the fact that Ethernet packets are sent under software control. The blue graph shows the TIC values while the red graphs show the WR software \(t_1\) and \(t_{4p}\) values. Both overlay and their constant (small) difference as indicated by the red arrows is only seen when zooming in.

![Figure 4: Example of Time Interval Counter and WR \(t_1\), \(t_{4p}\) measurements. Here the absolute values vary between 60-130 \(\mu\)s due to the embedded software which is non-deterministic. The zoomed area with the TIC\(_{t1}\) and TIC\(_{t4p}\) red arrows show the constant differences as expressed in equations (7) and (8).](image)

### 2.4. Oscilloscope measurement

A real time Digital Sampling Oscilloscope (DSO) is used to determine the time between the helper timestamp signal abscal_txts and the moment the Message Timestamp Point (MTP) of the Tx packet traverses the electrical Tx interface.

The Message Timestamp Point is defined as the beginning of the first symbol after the Start of Frame Delimiter (SFD, clause 7.3.4.1 of IEEE-1588-2008 [1]). The first symbol after the SFD is the MAC destination address that isn’t a fixed value which makes it difficult for triggering or signal recognition.

A good candidate for trigger recognition is the combination of the last octet of the preamble plus the SFD, which are coded in 1000BASE-X as \(<\text{D21.2}>\text{<D21.6}>\) (see also [8] clause 3.2, 4.2.6, and 36.2.4.5). The 8B/10B coded pattern for both code groups is the same for both positive and negative running disparity.

The message time stamp point follows immediately after the fixed sequence shown below:

\(\ldots\text{<D21.2>><D21.6>\ldots}\)

---

\(^1\) 180 measurements correspond to 3 minutes measurement time.
The delay between the helper timestamp signal abscal_txts and the bit sequence above (see Figure 5) can be determined with high accuracy using cross correlation and interpolation techniques [9]. The helper timestamp signal abscal_txts and the electrical Ethernet packet signal are sampled by a high speed real time oscilloscope. Figure 6 shows an example of a measurement for a Tx packet SFD. Two cross correlations are made. First the sampled waveform for the helper timestamp signal abscal_txts is correlated with the mathematical representation of a 16 ns “pulse”\(^2\). Second, the sampled waveform of the Ethernet packet is cross correlated with a mathematical representation of a 20 bit “Preamble-SFD” sequence as shown in Figure 5. Each cross correlation shows a maximum at a certain sample offset (see for an example Figure 7). The precision of the determination of the maximum can be further enhanced to pico-second level using interpolation.

The DSO measurement is the difference between the two maxima found which is the delay between the helper timestamp signal abscal_txts and the start of the SFD (see equation (9)).

\[
DSO = abscal_{txts} \rightarrow SFD
\]  

(9)

As can be seen in Figure 5, the cross correlation maximum found for the SFD has an offset with respect to the message timestamp point (MTP) which is half of the correlation sequence length used to correlate the preamble-SFD which is 20 bits in this case. The correlation offset is 10 bits multiplied by the line-speed\(^3\) of 1000BASE-X which results in 8 ns, so

\[
DSO_{abscal_{txts} \rightarrow MTP} = DSO + 8ns.
\]  

(10)

---

\(^2\) TIC measurements are threshold based rising edge measurements while the DSO measurement is based on cross correlation. Delay determination using cross correlation is the measure of two signals as a function of displacement of one relative to the other. This means that cross correlation is based on rising- and falling-edge. This may lead to a small delay determination error for \(t_o\) that should be taken into account for high accuracy calibration. Future work to enhance electrical absolute calibration should take this into consideration.

\(^3\) 1250 Mb/s corresponds to 800 ps/bit
Figure 6: Example of an oscilloscope measurement (i.e. waveform) for abscal_txts and Tx packet SFD. Note that abscal_txts lags the moment the actual Tx packet SFD traverses the electrical time reference plane. This is due to pipelining in the FPGA. In this picture the time from abscal_txts to Tx packet SFD is a negative number.

Figure 7: Cross correlations results for the abscal_txts and the SFD signal are shown. Left an overview and right, zoomed in. The arrows point to the correlation maxima. The delay between abscal_txts and SFD is the delay between the correlation maxima in DSO samples.

2.5. Combining the measurement results

The previous chapters described how to use a TIC and DSO to determine the time between the PPS and the helper timestamp signal abscal_txts (2.3) and the time between abscal_txts and the traversing of the Message Timestamp Point (MTP) over the electrical interface (2.4). Once these values are known then the absolute calibration constants can be calculated by combining equations (5)-(8), (9)-(10) into:

$$\Delta_{TXcal} = TIC_{t1} + (DSO + 8ns).$$  \hfill (11)

$$\Delta_{RXcal} = TIC_{t4p} + (DSO + 8ns) + t_{loop}.$$  \hfill (12)
3. **Equipment requirements**

3.1. **Check absolute calibration capability**

A WR device must have a helper timestamp signal abscal_txts output to perform electrical absolute calibration measurements. Legacy WR devices may not have this signal available. Check if you run the proper WR software version which should be capable of operating the WR device in “mode abscal”.

3.2. **Equipment list**

- Time Interval Counter (Keysight 52230A)
- Fast Real Time Digital Sampling Oscilloscope (LeCroy Waverunner 8254M or Keysight DSO-S-254A, see also Appendix A)
- External 10MHz reference clock and PPS. A Waveform Generator can be used (Keysight 33600A).
- SFP+ timing calibration module [10].
- Two, wide bandwidth, low phase unbalance, power splitters (Mini Circuits ZFRSC-123+) [11].
- Phase stable SMA-SMA coaxial cables.

Python scripts are used to perform electrical absolute calibration. These scripts work with the measurement equipment suggested above. Other drivers are necessary when measurement instruments of other instrument- or vendor-families are used.

3.3. **Software**

- Python 2.7 or 3.x
- Python VXI-11 which provides a pure python VXI-11 driver for controlling instruments over Ethernet. Download here [12] and
  
  ```bash
  $ python setup.py install
  ```
- Python SciPy (including numpy, scipy, matplotlib). Download here [13] and follow the installation instructions provided.
- Absolute calibration scripts:
  
  ```bash
  $ git clone git://ohwr.org/white-rabbit/wr-calibration.git
  $ cd wr-calibration
  $ git checkout master
  ```
  
  Absolute Calibration python scripts can be found in subdirectory:
  
  ```bash
  $ cd sw/abscal_scripts
  ```
4. Electrical Absolute Calibration Procedure

4.1. 10MHz Reference Clock

It is a pre-requisite that all measurement devices, TIC, DSO and WR device to be calibrated, share the same stable 10 MHz external timing reference (refer to Figure 8).
The WR device to be calibrated must be operated in mode abs. In this mode the WR device is locked to the external 10 MHz and PPS, similar to mode grandmaster (mode gm).
Use a Waveform Generator (for example Keysight 33600A) to create a 10MHz reference clock and PPS or use your GPS disciplined oscillator (GPSDO).

4.2. Connect the TIC and DSO to Ethernet

The TIC and DSO are controlled by python scripts using VXI-11 [12]. Make sure your instruments have an IP address and Subnet Mask and they are connected to a network that can be controlled by your computer. Connectivity can be checked by reading the identifier of the instrument.
Open a terminal window and start:

$ python
In [1]: import vx11
In [2]: scope = vx11.Instrument("<IP address>")
In [3]: scope.ask("*IDN?")

Should return something like:

KEYSIGHT TECHNOLOGIES, DSOS254A, MY55160101, 05.50.0004
4.3. Loopback Tx to Rx electrical interface

Put a calibrated loopback device in place between the Tx and Rx electrical interface. Usually the electrical interface will be the electrical connector of the SFP module. In this case the SFP+ timing calibration module [10] (see Figure 9) should be used. This module provides stable and repeatable access to the electrical time reference plane of the SFP connector. The serial data stream that crosses this time reference plane is probed without seriously affecting the loopback signal integrity. The probed signal is fed to a differential limiting amplifier and forwarded to the “P” and “N” connectors (see Figure 10). The differential signals must be connected via equal length phase stable SMA-SMA coaxial cables to the DSO (python scripts use channel 3 and 4 respectively to calculate the differential signal: Ch3–Ch4).

![Figure 9: SFP+ timing calibration module](image)

![Figure 10: Block schematic of the SFP+ Calibration module](image)

Each individual SFP+ timing calibration module contains three propagation delay calibration constants that are stored in the modules I2C EPROM [14]. The calibration constants are:

\[
\begin{align*}
SFP_{(Tx\rightarrow Out)} & \quad (13) \\
SFP_{(Rx\rightarrow Out)} & \quad (14) \\
t_{loop} & = SFP_{(Tx\rightarrow Rx)} \quad (15)
\end{align*}
\]

4.4. Measure the skew of your measurement setup

Chapter 2.3 and 2.4 describe two measurements to be performed. First, the time difference between PPS and the helper timestamp signal abscal_txts must be measured using a TIC. Second, the time difference between the helper timestamp signal abscal_txts and the moment a Ethernet Tx packet traverses the electrical interface must be measured using a DSO. This means that the helper timestamp signal abscal_txts must be split and be fed to the TIC as well as the DSO. The split is done using a wide bandwidth, low phase unbalance, power splitter (B in Figure 11) which will be connected with equal length cables to TIC In2 and DSO Ch1 (note: for high accuracy calibration delay unbalance needs to be characterized). Inevitable there are delays in the cables and instruments that lead to skew. In order to define a time reference plane at the connectors as specified in MIL-STD-348B, two skew measurements need to be done. Care is taken not to change the setup: the coax cables and
splitter B are part of the system that will be needed for the actual absolute calibration measurement.

Another wide bandwidth, low phase unbalance, power splitter (A in Figure 11) is used to split a single signal source into two reference signals. For the sake of simplicity the unbalance in the power splitters is not described in detail in this document. For high accuracy calibration the power splitters must be characterized and their delay unbalance must be taken into account.

The first measurement defines the TIC skew (see 4.4.1). Next the DSO skew will be measured (see 4.4.2).

![Figure 11: Skew Measurements]

**4.4.1. TIC Skew**

To measure the TIC skew, split a signal in two reference signals using power splitter A. Connect TIC In1 (that is to be the PPS signal in the calibration setup) to one of the reference signals and connect the input of power splitter B (that is to be the abscal_txts signal in the calibration setup) to the other reference signal as shown in the top half of Figure 11.

Output abscal_txts on your WR device can be used as input for power splitter A such that it serves as a pulse generator for the skew measurements. When doing so, make sure that your WR device runs “mode abscal”, is locked to the external 10 MHz clock/PPS and PTP is started (see 4.1).
Subdirectory `sw/abs_cal_scripts` contains the python scripts that do the measurement (refer to chapter 3.3). In order to gather some statistics, 600 measurements\(^4\) are made and stored in a file in subdirectory `data`.

A proper trigger level is important for the timing measurement. These levels can be set via input parameters `-i1` and `-i2` of the python script that controls the TIC. In this document a SPEC [4] and DIO [5] are electrical absolute calibrated as an indivisible entity. The `abs_cal_tcts` signal at LEMO output 3 of the DIO has an amplitude of 2.4V. MiniCircuits splitters ZFRSC-123+ have \(\sim9.75\) dB attenuation (\(\sim\) factor 3). A 2.4 V signal results in 0.8 V when split once and 260 mV when split twice. The trigger level is chosen to be 50% of the input signal level. The trigger levels are set to 0.4V and 0.13V for In1 and In2 respectively (see top half of Figure 11). The python script also terminates the TIC inputs to 50 ohm. Run the following python script that instructs the TIC to do 600 measurements:

```
$ python ../lib/Keysight_53230A.py <TIC IP address> -i1 0.4 -i2 0.13 -m 600
```

![Figure 12: Example output of TIC skew measurement](image)

The TIC measurement file automatically got a filename containing date and time (data/180803_12_07_21_freq_cnt_keysight_53230A in Figure 12). This file can be analysed using:

```
$ python tic_wr.py -tic_file <filename> -type tic
```

The result is plotted in a histogram (see Figure 13). TIC\(_{skew}\) mean and standard deviation values can be read from the terminal and are shown in the histogram.

\(^4\) 600 measurements correspond to 10 minutes measurement time. Another number may be chosen to reduce or increase measurement time (statistical significance will change accordingly).
4.4.2. DSO Skew

There are two SMA cables “P” and “N” that will connect to the SFP+ timing calibration module outputs “P” and “N” during calibration. To measure the DSO skew, disconnect the cable that leads to TIC In1 (see top half of Figure 11) and connect the cable that leads to DSO Ch3 (disconnect cable “P” from the SFP+ timing calibration module, see lower half of Figure 11).

Output abscal_txts on your WR device can be used as input for power splitter A such that it serves as a pulse generator for these measurements. When doing so, make sure that your WR device runs “mode abscal”, is locked to the external 10 MHz clock/PPS and PTP is started (see 4.1).

Subdirectory `sw/abscal_scripts` contains the python scripts that do the measurement (refer to chapter 3.3). Run the following python script that instructs the DSO to take 600 measurements and store them in waveform files in subdirectory `data`.

The trigger level of the oscilloscope is of less importance since cross correlation is used on channel 1 and 3 waveforms to determine delay. The python script initializes the DSO inputs to 50 ohm termination and set the channel 1 trigger level to 0.15V.

```
$ python edge_edge.py <DSO IP address> -m 600
```

The result is plotted in a histogram (see Figure 14). The second plot (Figure 15) shows the correlation peak, just for reference. DSO skew mean and standard deviation values can be read from the terminal and are shown in the histogram.
For the sake of simplicity the delay difference in the “P” and “N” SMA cables is not taken into account in this document. For high accuracy calibration the effect of cable difference on the delay measurement must be characterized and taken into account.

4.5. Measuring $t_A$

In paragraph 4.4 it was described how to obtain a measurement setup with known skews. These skews must be taken into account, thus moving the measurement time reference planes to the SMA connectors. The SMA connectors should be connected to the time reference planes of your WR device under calibration as shown in Figure 16. The measurements in this document were performed on a SPEC [4] (v4 hardware, wrpc-v4.2 firmware [15] and wrpc-sw->proposed_master based on wrpc-v4.2 software [16]) plus DIO [5].

As explained in paragraph 2.2, measuring $t_A$ is a two-step proces. The paragraphs below show how each measurement should be performed.
4.5.1. TIC measurement: PPS to abscal_txts

Both measurements, TIC and WR timestamps, must be done at the same time. The WR device must be put in mode abscal and locked to the external 10MHz and PPS reference. Verify that the WR device under calibration is connected as shown in Figure 16. The varying time between PPS and abscal_txts is measured and logged by the TIC. Each measurement corresponds to WR software timestamps t₁ and t₄p that are logged on the WR console (make sure that your terminal emulator creates a log file!).

A proper trigger level is important for the timing measurement. These levels can be set via input parameters –i₁ and –i₂ of the python script that controls the TIC. In this document a SPEC [4] and DIO [5] are electrical absolute calibrated as an indivisible entity. The abscal_txts signal at LEMO output 3 of the DIO has an amplitude of 2.4V. MiniCircuits splitters ZFRSC-123+ have ~9.75 dB attenuation (~ factor 3). A 2.4 V signal results in 0.8 V when split. The trigger level is chosen to be 50% of the input signal level. The trigger levels are set to 1.2V and 0.4V for In1 and In2 respectively (see Figure 16). The python script also terminate the TIC inputs to 50 ohm.

Stop PTP on the WR console:

```
# mode abscal
# ptp stop
```

Run the following python script that instructs the TIC (that has a particular IP address; refer to 4.2). First start the TIC measurement to take 600 measurements

```
$ python ../lib/Keysight_53230A.py <TIC IP address> -i1 1.2 -i2 0.4 -m 600
```

Next, start PTP on the WR console:

```
# ptp start
```

The above procedure makes sure that both measurements are in sync.

When the measurement completes, stop PTP again:

```
# ptp stop
```

Select the WR timestamps, that were logged during the measurements, from the terminal emulator log file (see Figure 17) and copy/paste them in a new file: data/wr_t1.

Three notes:

1. Discard the first recorded timestamp since this was the start of the TIC. The first recorded time interval measurement is made using the second timestamp.
2. File `data/wr_t1` may contain more entries than the measurement number recorded by the TIC, the latter is leading.
3. You may want to extend the name `data/wr_t1` with a date and time marker such that you can save the file for future reference.

![Figure 17: WR console timestamp selection. Columns show recorded t₁ (seconds, nanoseconds, phase), t₄ (seconds, nanoseconds, phase) and time difference (t₄₋t₁)](image)

Use the python script below to calculate the time difference between the TIC measurements and the WR timestamps. The TIC measurement file automatically got a filename (displayed on the terminal window; see example Figure 12) containing date and time. Use this file, together with the WR timestamps as input for:

```
$ python tic_wr.py -tic_file <TIC-filename> -wr_file data/wr_t1
```

Figure 18 shows all TIC and WR timestamp measurements. Figure 19 and Figure 20 show the histograms of the time TIC₁₁ and TIC₄₄ (see equations (7) and (8)). TIC₁₁ and TIC₄₄ mean and standard deviation values can be read from the terminal and are shown in the histograms.
Both data sets in Figure 18 should overlay which is a cross check that the TIC measurements were in sync with the WR timestamps. Figure 21 zoomed in on Figure 18 to verify that this is the case. Beware! Figure 22 shows the result when measurements are out of sync; a situation that leads to wrong results! In an out-of-sync situation, please check your start up procedure and/or timestamp selection described above.
4.5.2. DSO measurement: abscal_txts to Tx packet SFD
Verify that the WR device under calibration is connected as shown in Figure 16. The constant time between the helper timestamp signal abscal_txts and the Message Timestamp Point (MTP) on the electrical interface is measured using the SFP+ timing calibration module [10] (see picture in Figure 9) and a DSO.
The WR device must be put in mode abscal and locked to the external 10MHz and PPS reference. It is best to perform the “abscal_txts to Tx packet SFD” measurement after the “PPS to abscal_txts” measurement (described in paragraph 4.5.1) without re-synchronisation of the WR link (i.e. mode should already be mode abscal).
Start PTP on the WR console:

    # ptp start

DSO measurement is done using a python script. Currently LeCroy Waverunner 8254M or Keysight DSO-S-254A are supported by the script that expects channel connections as shown in Figure 16. Other oscilloscopes can be used as well but need a driver to readout their waveform memory (see Appendix A).
Start a single DSO measurement:

    $ python edge_sfd.py <DSO IP address> -timebase 50e-9

Executing the script results in three plots:

1. Correlation: Pulse & SFD
2. Waveforms: Ch1 & (Ch3-Ch4)
3. Histogram delay Pulse to SFD
For the time being only “Correlation: Pulse & SFD” plot is important.
The procedure to detect the proper cross correlation peaks is not left over to the software but might be automated in the future. Currently the proper cross correlation peaks are selected manually. Zoom in (pan button and left-, right-mouse button) and find the sample numbers for the maximum of the SFD and Pulse correlation peaks. The “Correlation: Pulse & SFD” plot should look like Figure 7. Examples of zoomed details are shown in Figure 23 and Figure 24.
The correlation maximum depends on the particular measurement setup so it is mandatory to determine these values before starting a real measurement. Close the plot windows and again take one measurement. This time the sample numbers of the estimated correlation maxima are given as command line parameters. In this example these would be `-edge 50` and `-sfd -14860`:

```bash
$ python edge_sfd.py <DSO IP address> -timebase 50e-9 -edge 50 -sfd -14860
```

Never mind the three plots that will again pop up. Look in the terminal window for the delay value that was calculated based on the estimated maximum correlation peaks. In example Figure 25 the delay is -74.6 ns. Don’t be surprised by a negative number since this means that (due to pipelining in the FPGA) the abscal_txts pulse lags the actual Tx packet SFD.
Close the plots. Next, 600 measurements can be acquired feeding the script with an estimated delay value\(^5\) and a delay tolerance (take 10 ns to discard outliers: \(-\text{tol} \ 10e-9\)).

\[
\$ \ \text{python \ edge\ sfd.py \ <DSO \ IP \ address>} \ -\text{timebase} \ 50e-9 \ -\text{edge} \ 50 \ -\text{sfd} \ -14860 \ -\text{delay} \ " -7.45536661500981e-08" \ -\text{tol} \ 10e-9 \ -m \ 600
\]

Now the plotted histogram is meaningful (see Figure 26). DSO mean and standard deviation values can be read from the terminal and are shown in the histogram. The value shown is the delay between abscal_txts and Tx packet SFD. Remember this value needs to be corrected for the 8 ns offset as is explained in paragraph 2.4 and equation (10). This will be taken into account when the final calculation is done (see 4.6).

![Figure 26: delay DSO = abscal_txts->SFD](image)

### 4.6. Bringing it all together

The actual \(\Delta_{TXcal}\) and \(\Delta_{RXcal}\) can now be calculated since all separate calibration constants are known or measured. Table 2 gives an overview of these parameters with references to their description in this document.

<table>
<thead>
<tr>
<th>Calibration parameter</th>
<th>Equation</th>
<th>Described in paragraph</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{loop} = SFP_{(Tx-to-Rx \ Dly)})</td>
<td>(15)</td>
<td>4.3</td>
</tr>
<tr>
<td>(SFP_{(Tx-to-Out \ Dly)})</td>
<td>(13)</td>
<td>4.3</td>
</tr>
<tr>
<td>(TIC_{skew})</td>
<td></td>
<td>4.4.1</td>
</tr>
<tr>
<td>(DSO_{skew})</td>
<td></td>
<td>4.4.2</td>
</tr>
<tr>
<td>(TIC_{t1})</td>
<td>(7)</td>
<td>4.5.1</td>
</tr>
<tr>
<td>(TIC_{t4p})</td>
<td>(8)</td>
<td>4.5.1</td>
</tr>
<tr>
<td>(DSO)</td>
<td>(9)</td>
<td>4.5.2</td>
</tr>
</tbody>
</table>

Table 2: References to equations and paragraphs that describe the calibration parameters

\(^5\) You might bump into python issue [https://bugs.python.org/issue22672](https://bugs.python.org/issue22672) “float arguments in scientific notation not supported by argparse” when entering scientific notation for delay. Note that for this reason parameter delay must be given with quotes and a leading space (i.e. \(-\text{delay} \ " \ < \text{estimated \ delay} >"\). See also [https://stackoverflow.com/questions/9025204](https://stackoverflow.com/questions/9025204).
TIC\textsubscript{skew} and DSO\textsubscript{skew} need to be taken into account, as well as the propagation delay from the electrical time reference plane of the SFP+ timing calibration module to its SMA connectors as specified in MIL-STD-348B.

Taking the above skews and delay into account, equations (11) and (12) can be rewritten into:

\[
\Delta_{TX} = (TIC_{t1} - TIC_{skew}) + 
(DSO + 8\text{ns} - DSO_{skew} - SFP_{Tx-to-OutDly})
\]

(16)

\[
\Delta_{RX} = (TIC_{t4p} - TIC_{skew}) + 
(DSO + 8\text{ns} - DSO_{skew} - SFP_{Tx-to-OutDly}) + t_{loop}.
\]

(17)

Finally, the results can be verified by

\$ python tic_wr.py -wr_file data/wr_t1 -type t1_4

which gives the mean and standard deviation values of the difference between the \(t_1\) and \(t_{4p}\) timestamps that were recorded by WR (also shown in the last column on the WR console, see Figure 17). The timestamp difference minus the calibrated loop delay \((t_{loop})\) should be equal to \(\Delta_{TXcal} - \Delta_{RXcal}\), refer to equation (3).

4.7. Apply the calibration constants

The current WR embedded software has placeholders for \(\Delta_{TX}\) and \(\Delta_{RX}\). As explained in paragraph 2.1 and referring to equation (3), these parameters can be assigned with the values calculated in equation (16) and (17)

\[
WR \ placeholder \ for \ \Delta_{TX} = \Delta_{TXcal}.
\]

(18)

\[
WR \ placeholder \ for \ \Delta_{RX} = -\Delta_{RXcal}.
\]

(19)
5. Appendix A

Guide to use other oscilloscopes

It is not mandatory to use the proposed Oscilloscopes (LeCroy Waverunner 8254M or Keysight DSO-S 245A). The real time digital oscilloscope should have enough analog bandwidth (at least 1.5 GHz) and a sampling speed in the order of 20 GSa/s such that cross correlation can be effective.

There are several functions that are called by the main python scripts. When the function interfaces are kept the same then one can write functions for other oscilloscopes as well. Below follows a description per function.

get_waveforms(scope, channels=[1,2,3,4],num_avg=1,output_dir="data")

scope -- instance of python-vxi connected to the oscilloscope
channels -- channels that are going to be measured for example '1,2'
num_avg -- the number of averages taken by the oscilloscope

This function performs an oscilloscope data acquisition. The raw waveform data is stored in the output_dir directory, in a file with the oscilloscope name and timestamp. This function returns the filename and an array with the raw waveform data.

the file output format is as described below:

"preamble"  data is a comma separated string describing the measurement parameters
"waveform_data"  is binary data

file_to_waveform(filename)

filename -- source file (as created by get_waveforms) from which to retrieve data.
This function converts a raw waveform file into a python <type 'dict'> with keys to a <type 'dict'> for each channel number, each of which is again a <type 'dict'> with keys:

- 'byte_order' : <type 'str'>
- 'preamble' : <type 'dict'>
- 'waveform' : <type 'numpy.ndarray'>

Each preamble is a <type 'dict'> with keys:

- 'acquisition_mode' : <type 'str'>
- 'bandwidth_maximum' : <type 'float'>
- 'bandwidth_minimum' : <type 'float'>
- 'completion' : <type 'int'>
- 'count' : <type 'int'>
- 'coupling' : <type 'str'>
- 'date' : <type 'str'>
- 'format' : <type 'str'>
- 'frame_model_#' : <type 'str'>
- 'points' : <type 'int'>
- 'time' : <type 'str'>
- 'type' : <type 'str'>
- 'x_display_origin' : <type 'float'>
- 'x_display_range' : <type 'float'>
- 'x_increment' : <type 'float'>
- 'x_origin' : <type 'float'>
- 'x_reference' : <type 'float'>
- 'x_units' : <type 'str'>
- 'y_display_origin' : <type 'float'>
- 'y_display_range' : <type 'float'>
- 'y_increment' : <type 'float'>
- 'y_origin' : <type 'float'>
- 'y_reference' : <type 'float'>
- 'y_units' : <type 'str'>

An example waveform_data dict looks like this:

```python
{1: {'byte_order': 'LSBFIRST',
     'preamble': {'y_display_origin': 30707.0,
                  'bandwidth_minimum': 0.0,
                  'y_units': 'Unit Name = V',
                  'x_units': 'Unit Name = S',
                  'time': '13:50:02'},
     'waveform': '\x8e\ef\8b\.....'},
  2: {'byte_order': 'LSBFIRST',
      'preamble': {'y_display_origin': 30707.0,
                   'bandwidth_minimum': 0.0,
                   'y_units': 'Unit Name = V',
                   'x_units': 'Unit Name = S',
                   'time': '13:50:02'},
      'waveform': 'C\x90X\x90\x90N\x90@\x90G\xa.....'}
}
preamble_string_to_dict(preamble_string)

This function is called by file_to_waveform and it converts the oscilloscopes preamble string into a format of `<type 'dict'>` that is used by file_to_waveform.

raw_to_scipy_array (waveform_raw, byte_order, preamble)

This function is called by file_to_waveform it converts the raw waveform block of data according to the preamble data. The waveform binary data is scaled to an x_data, y_data array using the scale units given in preamble.

```
preamble     -- preamble dict
waveform_raw -- raw block datastream excluding length header
byteorder    -- byteorder (MSBFIRST, LSBFIRST) of the BINARY, BYTE and WORD formats.

returns: <type 'numpy.ndarray'>
array([[x1,x2,x3,...,xn],[y1,y2,y3,...,yn]])
```

osc_init(scope, time_base = 50.0e-9)

This is a function to initialize the oscilloscope and make it ready for acquisition. It enables channels, sets volt per division, coupling, trigger and external reference clock use.

```
scope      -- instance of python-vxi connected to the oscilloscope

time_base  -- <float> time base, default 50 ns/div
```

check_waveforms(waveform_data)

This function checks for the consistency of the captured waveform.

```
waveform_data -- <type 'dict'>
waveform_data (as returned by function "file_to_waveform")

returns: number of points (of the first waveform found)
```
6. Appendix B

Calibration of an FMC module.

When, for a particular WR application, the electrical time reference plane of the PPS signal is defined on an FMC (for example the SPEC [4] and DIO [5]) then absolute calibration must be done as described in the main document, taking the cards as one indivisible entity. If FMC exchange is foreseen then separate FMC calibration is necessary. Figure 27 shows how this could be done using a FMC calibration PCB.

![Figure 27: Three measurements to determine FMC delay.](image)

The calibration PCB powers the FMC under calibration and translates the FMC connector interface to the opposite gender of the connector that is used to forward the electrical time reference plane of the PPS signal on the FMC.

Connectors of opposite gender enables a delay differential measurement. The delay difference between delay measurement M1 and M2 is equal to the delay of the calibration PCB plus FMC.

The delay between the input connector on the calibration PCB and the open FMC connector is determined (M3) using differential Time Domain Reflectometry (TDR).

The electrical time reference plane of the PPS signal can now be transferred to the FMC connector since the delay between both time reference planes can be calculated with equation (20).

\[
\Delta_{\text{FMC}} = M1 - M2 - M3 \tag{20}
\]

The FMC delay can be stored in the FMC EPROM as a FMC calibration constant such that system management can access this parameter and automatically take it into account when a link is setup. This enables exchange of FMC cards without the need for recalibration.
7. Appendix C

PHY word width

Cross correlation on the helper timestamp signal abscal_txts is done with a mathematical representation of a “pulse”. For a proper cross correlation peak the width of the mathematical pulse should be the width of the sampled waveform pulse. Usually the width of the timestamp signal abscal_txts is 16 ns (sysclk = 62.5 MHz), corresponding to a PHY that takes 16 bits and serializes them into 20 bits per sysclk tick. The edge_sfd.py python script has a `bitwidth` switch (default = 20). If the sampled pulse width differs from 16 ns then the `bitwidth` switch should be set accordingly for a proper correlation result.

The helper timestamp signal abscal_rxts was used in the past but is no longer necessary for electrical absolute calibration. Table 3 below is added to this document just for reference. The `bitwidth` switch depends on the PHY PCS width.

<table>
<thead>
<tr>
<th>WR Device</th>
<th>FPGA</th>
<th>WRPC VHDL generic: pcs_16bit</th>
<th>Bitwidth abscal_txts</th>
<th>Bitwidth abscal_rxts</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPEC</td>
<td>Spartan-6</td>
<td>false</td>
<td>20</td>
<td>10</td>
</tr>
<tr>
<td>WR-Switch</td>
<td>Virtex-6</td>
<td>true</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>CLB (KM3NeT)</td>
<td>Kintex-7</td>
<td>true</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 3: `bitwidth` values for various WR devices / FPGA families
8. References

[16] wrpc-sw software, git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git