ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

THE DISTRIBUTED OSCILLOSCOPE: A LARGE-SCALE FULLY SYNCHRONISED DATA ACQUISITION SYSTEM OVER WHITE RABBIT

D. Lampridis*, T. Gingold, M. Malczak[1], F. Vaga,
T. Włostowski, A. A. Wujek, CERN, Geneva, Switzerland
[1]also at Warsaw University of Technology, Warsaw, Poland

Abstract

A common need in large scientific experiments is the ability to monitor by means of simultaneous data acquisition across the whole installation. Data is acquired as a result of triggers which may come either from external sources, or from internal triggering of one of the acquisition nodes. However, a problem arises from the fact that once the trigger is generated, it will not arrive to the receiving nodes simultaneously, due to varying distances and environmental conditions. The Distributed Oscilloscope (DO) concept attempts to address this problem by leveraging the sub-nanosecond synchronisation and deterministic data delivery provided by White Rabbit (WR) and augmenting it with automatic discovery of acquisition nodes and complex trigger event scheduling, in order to provide the illusion of a virtual oscilloscope. This paper presents the current state of the DO, including work done on the Field-Programmable Gate Array (FPGA) and software level to enhance existing acquisition hardware, as well as a new protocol based on existing industrial standards. It also includes test results obtained from a demonstrator used to showcase the DO concept, based on two digitisers separated by a 2.5 km optical fibre.

INTRODUCTION

From the monitoring of particle accelerators to smart electrical grids and from scientific experiments performed at the bottom of the sea to astronomical observatories and meteorological stations on mountaintops, a common requirement in large-scale Test and Measurement (T&M) setups has always been the ability to remotely control, automate and synchronise the involved equipment (instruments). This is of course true even in smaller setups, such as the ones found in laboratories, but it becomes even more important when the equipment is distributed across longer distances, or placed in remote and hard-to-reach locations.

The accelerator complex of the European Organisation for Nuclear Research (CERN), which includes the Large Hadron Collider (LHC) with a circumference of 27 km, is a prime example of such an installation. CERN operators need to be able to monitor the state of the accelerators and, very often, to correlate measurements performed across the accelerators (e.g. at beam transfer lines).

This paper presents the White Rabbit Trigger Distribution (WRTD) [1] system, a new development at CERN that allows sub-nanosecond synchronisation of instruments across several kilometres of distance and distribution of triggers over White Rabbit (WR) [2, 3] in the form of network messages. It also presents the Distributed Oscilloscope (DO) [4], a proof-of-concept project for WRTD.

* dimitrios.lampridis@cern.ch

BACKGROUND

Historically, the need for synchronisation of instruments has been addressed by various types of T&M systems. Already in the late 1960s, the General Purpose Interface Bus (IEEE-488, GPIB) [5] was introduced to allow control and readout of up to 14 daisy-chained instruments from a computer, using a cable of up to 20 meters total length. GPIB extenders were later introduced to overcome these limitations. Long coaxial cables were (and still are being) used to synchronise the instruments and distribute triggers by means of their external trigger input/output and "sync" ports. In the 1980s, the popularity of VMEbus led to the development of the VME eXtensions for Instrumentation (VXI) [6], and a decade later, PCI eXtensions for Instrumentation (PXI) [7] was added to the list. Both VXI and PXI included multiple trigger lines on their backplanes for synchronisation between the instruments.

In more recent years, the proliferation of Ethernet-based computer networks led in 2005 to the introduction of the LAN eXtensions for Instrumentation (LXI) [8]. Where VXI and PXI also imposed the mechanical format of the instruments, LXI focused on the protocol and provided services (such as automatic discovery of attached devices), allowing for rack-mounted, bench-top, modular or any other type of form factor to be used and interconnected, as long as they have a Local Area Network (LAN) port. Furthermore, the Ethernet network allowed for long distances between the instruments and their operators.

The LXI standard is divided into the so-called "Device Specification" [9] which contains requirements for all LXI-compatible devices, and a set of optional "Extended Functions". A group of three such extended functions, Clock Synchronisation (CS) [10], Event Messaging (EM) [11] and Timestamped Data (TD) [12] provide a synchronisation layer on top of the core LXI standard, offering similar capabilities to those found on the backplanes of VXI and PXI. LXI CS is based on the IEEE 1588 Precision Time Protocol (PTP) [13] and it specifies its own PTP profile. LXI devices supporting this function have their clocks synchronised with sub-microsecond accuracy. LXI EM defines the methods for the exchange of messages directly between the instruments using multicast User Datagram Protocol (UDP) or point-to-point Transmission Control Protocol (TCP) connections. LXI TD adds timestamps to all messages, events

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

and acquired data. The combination of LXI CS, EM and TD allows for events to be tied to absolute times for precise triggering, synchronisation and correlation among the instruments.

Nevertheless, for time-sensitive applications, the sub-microsecond accuracy provided by typical PTP implementations, such as the one in LXI, is often not enough. Modern oscilloscopes and digitisers provide the ability to take billions of samples per second, in order to be able to capture events that last only for a few nanoseconds (or less); in this context, even half a microsecond "off" in the trigger moment could render the measurement unusable or just impossible to correlate with measurements from other oscilloscopes on the same network.

To address this issue, CERN initiated in 2008 the White Rabbit project, an effort to develop a fully deterministic Ethernet-based network which improves on top of PTP by providing sub-nanosecond accuracy. Furthermore, during the first run of the LHC (2009-2013), the operators were witnessing transverse instabilities in the accelerator, which led to the development and deployment of the LHC Instability Trigger (LIST) [14, 15] distribution project during the first Long Shutdown (LS1). The LIST, which is still operational today, is a trigger distribution system based on WR. It can receive a trigger from a "cloud" of devices and distribute it to all relevant devices to for example freeze their acquisition buffers. This was used during the second run of the LHC (2015-2018) to synchronise instruments across the accelerator and help in the early detection of instability onsets.

In 2018, CERN launched the WRTD project, a new effort which builds upon the experience gathered from LIST and which aims to provide a generic event distribution system, one that could be easily integrated in various types of instruments and which could cater for a larger variety of experiments. Furthermore, and in an effort to bring it closer to existing standards (and eventually merge it with them), WRTD is based on the LXI CS, EM and TD extensions and it uses the same LXI event message format.

WRTD

In WRTD, Nodes receive "input" Events and distribute them to other Nodes over WR in the form of network Messages that are used to transfer the timestamp of the input Event. The receiving Nodes are programmed to execute some "output" Event (action) upon reception of a particular Message, potentially with some fixed delay added to the timestamp.

There are two main categories of WRTD applications:

1. A "source" Node receives an input Event, adds a fixed delay to its timestamp and distributes it to other Nodes. As long as the fixed delay added is greater than the upper-bound latency of the network (a fundamental feature of WR itself), all other Nodes will receive the Message before the programmed time and will execute simultaneously their action (thanks to the sub-nanosecond synchronisation provided by WR). This is the typical scenario for an event distribution system.

2. The receiving Nodes are recording devices (e.g. digitisers), capable of storing data in a recording buffer. The source Node transmits the Message, with or without a fixed delay added to its timestamp. When one of the destination Nodes receives the Message, it stops recording and rolls-back its buffer to the moment specified by the timestamp in the received Message (provided that it has a large enough buffer to compensate for the latency). Thus, all Nodes will deliver recorded data from the moment in the past when the input Event was originally received at the source Node. In particular, this is the way that the Distributed Oscilloscope uses WRTD.

Of course, the above list is not exhaustive, there are many other potential applications but they are usually permutations of one of the above scenarios.

Basic Concepts

This section introduces various basic concepts of WRTD in alphabetical order. These concepts are fundamental to understanding how WRTD works (and, by extension, how the DO works).

Alarm An Alarm is simply a user-defined moment to generate internally an input Event on a Node. Every Node checks periodically if any of the declared (and enabled) Alarms need to be triggered.

Event Events represent inputs and outputs of a WRTD Node. Input Events received on a Node will result in an output Event to be generated (assuming that the relevant Rule exists to associate an input to an output). In that sense, inputs Events are the causes, while output Events are the effects.

Input Events can originate from a Local Channel, an inbound network Message, or an Alarm. Output Events can be delivered to a Local Channel, or an outbound network Message.

An Event is essentially a combination of an Event ID (the "what") and an Event timestamp (the "when").

Local Channel Local Channels represent the connections of a Node to its environment. They can be either inputs or outputs.

A Local Channel input delivers input Events to the Node. Typical examples include the external trigger input of a digitiser, a Time to Digital Converter (TDC) or a TTL input channel on a digital I/O board.

A Local Channel output transmits output Events from the Node. Typical examples include a Fine Delay generator or a TTL output channel on a digital I/O board.

All Local Channels use specific -reserved- Event IDs. For inputs these take the form `LC-I<x>`, while for outputs

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

they take the form of LC-O<x>, where <x> is a number starting from 1 (e.g. LC-I1). All other Event IDs are considered to refer to network messages.

Message WRTD Event Messages (or, simply, Messages) follow the LXI Event Messaging format, as defined in Rule 4.3 of the LXI Event Messaging Extended Function specification. To ensure compatibility and interoperability with LXI devices, WRTD Event Messages are transmitted using multicast UDP on address 224.0.23.159, port 5044 (Rule 3.3.1 of the specification). Each Message is transmitted as a single Ethernet frame, with a UDP header and a payload as shown in Fig. 1.

Within a Message, the "Domain" and "Flag" LXI fields (octets 3 and 36 respectively) are fixed to zero. Although there are currently no "Data Fields" defined (octets 37 and beyond), it should be highlighted that the LXI Event message format supports an arbitrary number of data fields, in the form of Type/Length/Value (TLV) triplets, which could be used to provide additional functionality to WRTD in the future.

Node WRTD is made of Nodes, connected to each other over a WR network. Nodes receive input Events and send output Events.

Every Node has Local Channel inputs and/or outputs allowing it to interact with its environment. It also has a connection to a WR network, allowing it to send and/or receive Messages to other Nodes.

Rule In WRTD, the programming of Events, Messages and associated actions is done by defining Rules. A Rule simply declares a relationship between an input (cause) and an output (effect) Event. A Rule can state that when a specific Event is received a Message should be transmitted or, that when a Message is received an output Event should be generated.

An input Event to a Rule can be a Local Channel, a Message or an Alarm. When an input Event is received by a Node, WRTD tries to match it with any declared (and enabled) Rule. Once an input Event has been matched and all delays have been applied to it, it is forwarded to the next processing block that generates the preconfigured output Event, which is then delivered to a Local Channel or sent over the network.

Reference Nodes

WRTD provides so-called "reference" Nodes for the most common use-cases. These Nodes are based on open hardware designs, which are also commercially available from various manufacturers.

Every reference Node has a Field-Programmable Gate Array (FPGA) with one or more RISC-V soft-CPUs and a WR core [16]. The soft-CPUs (based on the Mock Turtle project [17]) run embedded software (firmware) which implements all the WRTD-related functions and connects WRTD to the local application, while the WR core provides the interface to send and receive Messages over the WR network.

Of course, every reference Node additionally includes application-specific cores to handle everything else that is not part of WRTD (e.g. a core to control an Analogue-to-Digital Converter).

A firmware development framework (built on top of the MT firmware framework) is also available. This framework, which is used by the provided reference Nodes, can accelerate the development of firmware for new reference Nodes and ensure full compatibility with WRTD.

Currently, the following reference Nodes are available:

1. A quad channel, 100 MHz, 14bit PCIe-based digitiser capable of triggering via WRTD (SPEC150T-FMC-ADC).

2. A pulse-in/pulse-out WRTD Node in VME format, for generic trigger distribution applications (SVEC-TDC-FD).

SPEC150T-FMC-ADC This is a WRTD Node based on the Simple PCIe FMC Carrier (SPEC[1]) [18] and the Fmc-Adc-100M-14b-4cha (FMC-ADC) [19] ditigiser. It provides the possibility to generate WRTD Messages based on trigger events of the FMC-ADC, as well as to trigger the FMC-ADC from incoming WRTD Messages.

It should be noted that the 100 MHz sampling clock of the FMC-ADC is not yet synchronised to WR. This means that there can be up to one sample period (10 ns) misalignment when correlating data from two SPEC150T-FMC-ADC Nodes. This is a limitation of the current FMC-ADC design, not of WRTD.

The architecture of the Node can be seen in Fig. 2. The user application running on the host communicates with the Node via the WRTD library for all WRTD-related aspects and via the ADC library [20] for everything else. Internally, the Node is running one firmware application, responsible for retrieving/delivering triggers from/to the FMC-ADC and the WR network. Since the ADC library and the firmware application access different and separate parts of the FMC-ADC, there is no danger of accessing the same resources simultaneously.

The Node exposes five Local Channel inputs and one output. Their mapping to ADC functions is shown in Table 1. The forwarding delays introduced by the Node[2] are summarised in Table 2.

[1] This Node does not use the standard SPEC, because the fitted FPGA (XC6SLX45T) is not large enough for the complete design. Instead it uses the pin-compatible XC6SLX150T FPGA (the rest of the board is exactly the same, only the FPGA chip is different). This special version is available from the manufacturers of the SPEC board upon request.

[2] The delays in Tables 2 and 3 are representative of real performance when the Node is programmed with a single Rule. If more Rules are declared, then the Node will need to perform more checks in every loop and/or might be executing a Rule when another Event arrives. In any case, WRTD provides the necessary diagnostics and statistics to calculate worst-case delays in such scenarios.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	0x4c ('L')								0x58 ('X')								0x49 ('I')								0x00							
4	32	Event ID [0:3]																															
8	64	Event ID [4:7]																															
12	96	Event ID [8:11]																															
16	128	Event ID [12:15]																															
20	160	Sequence number																															
24	192	Event timestamp seconds (32 lower bits)																															
28	224	Event timestamp nanoseconds																															
32	256	Event timestamp fractional nanoseconds																Event timestamp seconds (16 upper bits)															
36	288	0x00								0x00								0x00								0x00							

Figure 1: Contents of a WRTD Event Message.

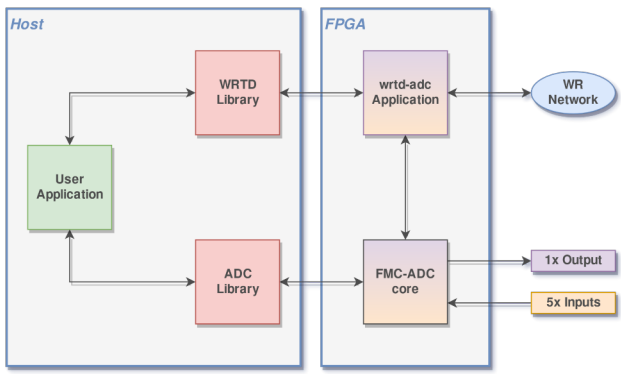

Figure 2: SPEC150T-FMC-ADC reference WRTD Node.

Table 1: Mapping of SPEC150T-FMC-ADC Functions to Local Channels.

Channel	Function
LC-I1	ADC Channel #1 internal trigger
LC-I2	ADC Channel #2 internal trigger
LC-I3	ADC Channel #3 internal trigger
LC-I4	ADC Channel #4 internal trigger
LC-I5	ADC external trigger
LC-O1	ADC auxiliary trigger

Table 2: Forwarding Delays of the SPEC150T-FMC-ADC.

Direction	Value
Trigger out to WR	12 µs
WR to trigger in	12 µs

SVEC-TDC-FD This is a WRTD Node based on the Simple VME FMC Carrier (SVEC) [21], the FMC Time to Digital Converter (FMC-TDC) [22] and the FMC Fine Delay generator (FMC-FD) [23].

The basic principle of this Node is simple: it takes in external pulses on its FMC-TDC inputs, timestamps them using WR time and converts them to WRTD Messages, to be sent over the WR network. Conversely, the Node also receives WRTD Messages which are then used to generate pulses at a predefined moment on one of the FMC-FD outputs. As such, it can be seen as a "pulse-to-message" and "message-to-pulse" converter with applications in the fields of pulse distribution, trigger synchronisation, etc.

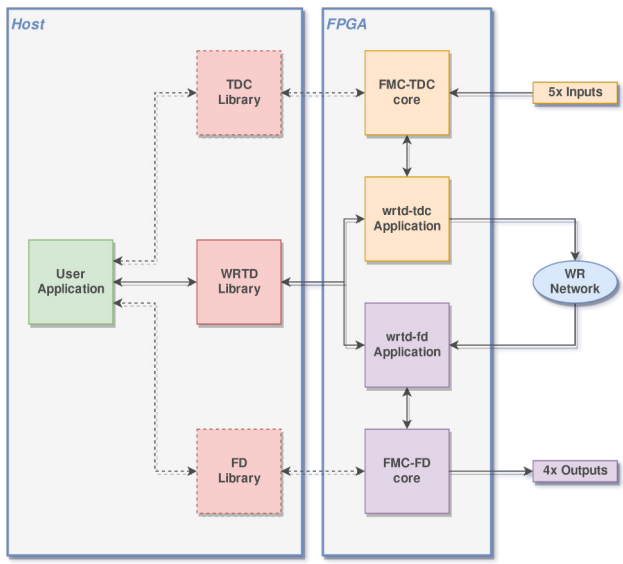

Figure 3: SVEC-TDC-FD reference WRTD Node.

The architecture of the Node can be seen in Fig. 3. The user application running on the host communicates with the Node via the WRTD library. Internally, the Node is running two firmware applications, one responsible for retrieving triggers from the FMC-TDC and delivering them to the WR network and the other for retrieving triggers from the WR network and delivering them to the FMC-FD. The user application can also use the FMC-TDC and FMC-FD libraries to fine tune the respective cores (e.g. to change the pulse length on FMC-FD outputs). However, this should be done with care, as it could lead to a race condition between the user and firmware applications.

The Node exposes five Local Channel inputs and four outputs. The inputs are mapped to the five channels of the FMC-TDC, while the outputs are mapped to the four channels of the FMC-FD. The forwarding delays introduced by the Node are summarised in Table 3.

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

Table 3: Forwarding delays of the SVEC-TDC-FD.

Direction	Value
Pulse in to WR	20 µs
WR to pulse out	40 µs

Provided Software

WRTD provides three options for accessing a Node via software (in decreasing order of flexibility and complexity):

1. a C library.
2. a Python wrapper for the C library.
3. a command-line tool built using the Python wrapper.

C Library The WRTD C Library is the standard, most flexible (but also complex) way of accessing a WRTD Node. The library Application Programming Interface (API) mimics that of an Interchangeable Virtual Instruments (IVI) [24] driver, with a strong influence from the IVI-3.2 "Inherent Capabilities" [25] and IVI-3.15 "IviLxiSync" [26] specifications.

```
struct wrtd_dev *wrtd;
uint32_t node_id = 1;
char error_description[256];
enum wrtd_status status;

status = wrtd_init(node_id, 0, NULL, &wrtd);

if (status != WRTD_SUCCESS) {
  wrtd_get_error(wrtd, &status, 256, error_description);
  fprintf(stderr, error_description);
  return 1;
}

status = wrtd_add_rule(wrtd, "rule1");

status = wrtd_set_attr_string(wrtd, "rule1",
WRTD_ATTR_RULE_SOURCE, "LC-I2");

status = wrtd_set_attr_string(wrtd, "rule1",
WRTD_ATTR_RULE_DESTINATION, "NET0");

status = wrtd_set_attr_bool(wrtd, "rule1",
WRTD_ATTR_RULE_ENABLED, true);
```

Listing 1: Example of declaring a Rule using the C library.

Listing 1 shows an example of accessing a Node using the C library and declaring a Rule. Please note that, for the sake of brevity, the example omits error-checking after the first function call.

Python Wrapper The WRTD Python wrapper provides a thin wrapper around the C Library, using the Python "ctypes" package. The wrapper is provided as a Python package with a single class (`PyWrtd`) that encapsulates the complete WRTD C Library.

The Python wrapper makes it easier to develop applications that make use of WRTD since it encapsulates all device initialisation and error handling within the provided class. Listing 2 shows the same operations as in Listing 1, being performed with the Python wrapper instead.

```
from PyWrtd import PyWrtd

wrtd = PyWrtd(1)
wrtd.add_rule('rule1')
wrtd.set_attr_string(
     'rule1', PyWrtd.WRTD_ATTR_RULE_SOURCE, 'LC-I2')
wrtd.set_attr_string(
     'rule1', PyWrtd.WRTD_ATTR_RULE_DESTINATION, 'NET0')
wrtd.set_attr_bool(
     'rule1', PyWrtd.WRTD_ATTR_RULE_ENABLED, True)
```

Listing 2: Example of declaring a Rule using the Python wrapper.

Command-line Tool WRTD provides a Python based tool (wrtd-tool) for accessing a Node. The tool implements several different operations on a Node. It supports most of the functionality provided by the Python Wrapper.

```
$ wrtd-tool add-rule 1 rule1
$ wrtd-tool set-rule 1 rule1 LC-I2 NET0
$ wrtd-tool enable-rule 1 rule1
```

Listing 3: Example of declaring a Rule using the command-line tool.

When the additional flexibility provided by the C and Python APIs is not required, the command-line tool offers the easiest and briefest way of accessing a Node. Listing 3 shows the same operations as in Listings 1 and 2, being performed with the command-line tool instead.

DISTRIBUTED OSCILLOSCOPE

The DO is built on top of WRTD as a set of Python modules. These include:

1. User applications
2. Server
3. Device applications

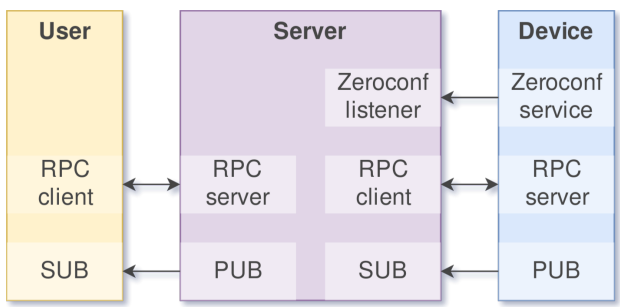

Figure 4: Inter-module communication in the DO.

Figure 4 shows the various communication channels between the three modules. Within a given network, there can

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

be many User and Device applications, but only one Server. Devices use Zeroconf to announce themselves to the Server. All synchronous communication is done via Remote Procedure Calls (RPC), using a "request/response" messaging scheme, while all other (asynchronous) communication, including data publishing and notifications is performed using a "publisher/subscriber" scheme. Both messaging schemes are implemented with ZeroMQ.

User Applications

User applications are clients with the ability to control the configuration of devices (WRTD Nodes) and to collect, post-process and -possibly- display the acquired data from them.

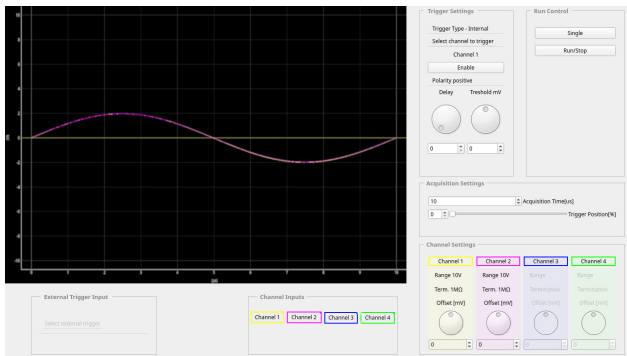

Figure 5: DO GUI user application made with PyQt.

Currently, there are two User applications available: the "GUI" and the "Testbench". The first is the main DO user application, which resembles a standard benchtop oscilloscope. All graphical elements of the GUI are built using PyQt. Figure 5 shows an impression of the GUI. The second is used to test the Server and the Device applications, as well as to perform diagnostics and calculate performance statistics.

Server

The Server is a central unit responsible for managing all the connections, preprocessing the data and providing a common interface for connected applications. It acts as a proxy between Device and User applications.

Device Applications

Device applications provide access to the underlying hardware devices (WRTD Nodes).

There is currently one Device application available, which works with the SPEC150T-FMC-ADC WRTD reference Node. It is built on top of the WRTD Python wrapper (for the WRTD-related operations on the Node, such as adding/configuring/enabling Rules) and ADC-LIB (for all other aspects, such as configuration of acquisition parameters).

Demonstrator

Figure 6 shows the laboratory demonstrator that was built in order to showcase the DO concept and measure its performance.

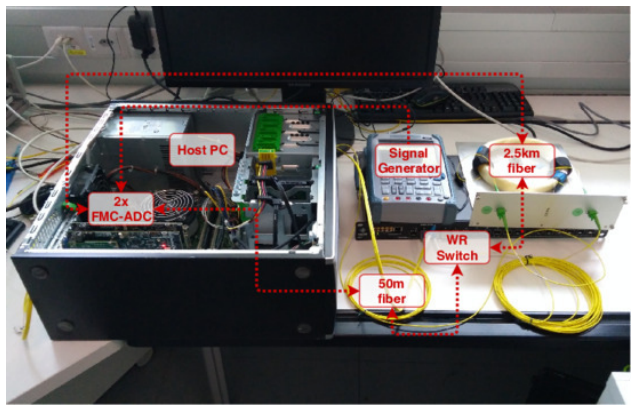

Figure 6: The DO demonstrator.

The demonstrator uses two SPEC150T-FMC-ADC reference Nodes, installed on two slots of the same computer. Analogue input channel #4 of each Node is connected to a signal generator using matched-length coaxial cables. Furthermore, the WR port of each Node is connected to a standard WR switch. In order to emulate a distributed acquisition setup, one Node is connected using a 50 m fibre, while the other is connected using a 2.5 km roll of fibre.

The total delay introduced when distributing triggers over this setup can be calculated as:

$$\Delta T_{SUM} = 2 * \Delta T_{WRTD} + (L_1 + L_2) * \Delta T_{fibre} + \Delta T_{WRS} \quad (1)$$

Where ΔT_{WRTD} is the delay introduced by the reference Node (both introduce 12 µs), L_1 and L_2 are the lengths of the two fibres (50 m and 2500 m), ΔT_{fibre} is the propagation delay for one meter of fibre (5 ns) and ΔT_{WRS} is the worst-case packet forwarding delay of a WR switch (10 µs). Putting these numbers in Eq. (1), we calculate 46.75 µs of total delay.

Given the fact the that sampling clock of the FMC-ADC is 10 ns, that it has four channels and that it uses 2 bytes per sample, the trigger-receiving Node needs to be able to "sacrifice" up to 18 700 samples (or 37 400 bytes) of memory in order to successfully roll back its buffer to the moment of trigger on the other Node. This is not an issue at all for the SPEC150T-FMC-ADC that is equipped with 256 MB of memory.

RESULTS

A series of measurements were performed using the DO demonstrator. The aim of the measurements was to evaluate the accuracy and precision of data alignment between two Nodes that receive exactly the same analogue signal.

For these measurements, a 1 kHz square-wave signal was provided on ADC channel #4 of both Nodes of the demonstrator, using two matched-length (both labelled as "8 ns")

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

coaxial cables from the signal generator. One of the Nodes was chosen as the trigger source, configured for internal positive edge triggering on channel #4 crossing the 0 V threshold. The acquisition (same on both Nodes) was configured with 50% pre/post sampling and a duration that was shorter than the period (1 ms) of the analogue signal. A measurement consisted of 20000 acquisitions performed with this setup.

As already mentioned, the sampling clock of the ADCs is not locked to WR, therefore the expected precision is 20 ns (equal to two periods of the sampling clock). Another factor to take into consideration is that the Nodes were running an early development version of the SPEC150T-FMC-ADC reference design, which did not support retrieving and applying the FMC-ADC calibration values. Therefore, the two FMC-ADCs were running uncalibrated.

The accuracy of data alignment between the two Nodes was calculated for each acquisition by detecting the zero-crossing of each data set and taking the distance between the two zero-crossing points. The zero-crossing was calculated by taking a linear function defined by the two adjacent samples of the data set that show a change of sign (from negative to positive) and calculating the value for which this function becomes zero.

The precision was measured from the standard deviation (σ) of the Gaussian distribution fitted to the measured accuracy. The resulting histogram of the measurement is shown in Fig. 7. As it can be seen from the results, the accuracy (μ) is 180 ps and the precision (6σ) is 20.1 ns.

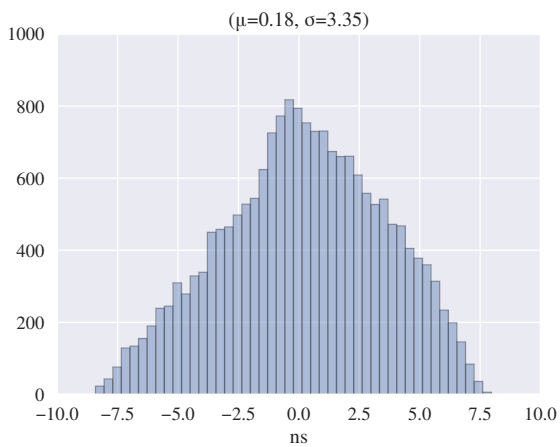

Figure 7: Histogram of data alignment measurement.

In order to verify that the lengths of the cables used to deliver the signal to the two Nodes were indeed matched, a second measurement was performed by exchanging the two cables (and keeping everything else the same). The resulting histogram of the measurement is shown in Fig. 8.

As expected, the precision remains the same. However, the fact that the mean value is different shows that the two cables do not have exactly the same propagation delay. We can calculate their signal propagation difference by subtracting the two mean values and dividing by two, to arrive to a mismatch of 435 ps.

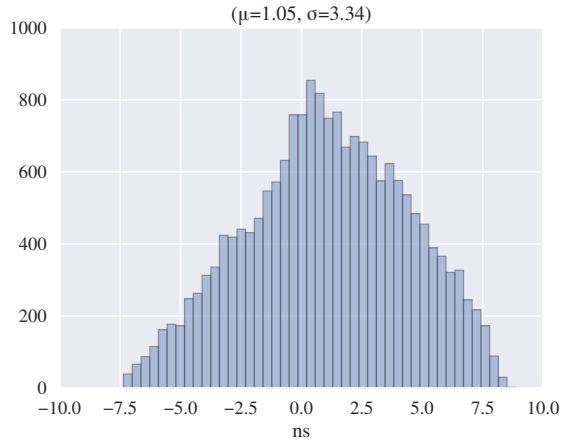

Figure 8: Histogram of data alignment measurement with reversed signal cables.

CONCLUSION

The Distributed Oscilloscope successfully demonstrates the use of WRTD for creating a virtual oscilloscope application, able to correlate signals across several kilometres of distance. The generic API and open design of WRTD itself, as well as its reliance on existing industrial standards and the availability of good documentation, provide a solid base for building many other kinds of trigger distribution systems with it.

In the future, it is foreseen to try to merge WRTD with the IVI/LXI standards. Such a development would allow instruments with an IVI driver (e.g. PXI, LXI instruments) and a WR interface to exchange events with each other. It would also enable the production of commercial WRTD-enabled devices.

REFERENCES

[1] *WRTD* project page, https://ohwr.org/project/wrtd/wikis/home

[2] *White Rabbit* project page, https://ohwr.org/project/white-rabbit/wikis/home

[3] J. Serrano *et al.*, "The White Rabbit Project", in *Proc. 12th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'09)*, Kobe, Japan, 2009, paper TUC004.

[4] Distributed Oscilloscope project page, https://ohwr.org/project/distributed-oscilloscope/wikis/home

[5] *IEEE 488 - IEEE Standard Digital Interface for Programmable Instrumentation*, Institute of Electrical and Electronics Engineers, 1987, ISBN 0-471-62222-2.

[6] VXIbus Consortium, http://www.vxibus.org

[7] PXI Systems Alliance, http://www.pxisa.org

[8] LXI Consortium, http://www.lxistandard.org

ISBN: 978-3-95450-209-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2019-TUBPR01

[9] *LXI Device Specification 2016*, LXI Consortium, revision 1.5.01, 14 March 2017.

[10] *LXI Clock Synchronization Extended Function*, LXI Consortium, revision 1.0, 8 November 2016.

[11] *LXI Event Messaging Extended Function*, LXI Consortium, revision 1.0, 8 November 2016.

[12] *LXI Timestamped Data Extended Function*, LXI Consortium, revision 1.0, 8 November 2016.

[13] *IEEE 1588 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, Institute of Electrical and Electronics Engineers, 2008, ISBN 978-0-7381-5400-8.

[14] *LIST* project page, https://ohwr.org/project/list/wikis/home

[15] T. Wlostowski *et al.*, "Trigger and RF Distribution Using White Rabbit", in *Proc. 15th Int. Conf. on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15)*, Melbourne, Australia, 2015, paper WEC3O01, pp. 619–623.

[16] *White Rabbit PTP Core* project page, https://ohwr.org/project/wr-cores/wikis/wrpc-core

[17] *Mock Turtle* project page, https://ohwr.org/project/mock-turtle/wikis/home

[18] *SPEC* project page, https://ohwr.org/project/spec/wikis/home

[19] *Fmc-Adc-100M-14b-4cha* project page, https://ohwr.org/project/fmc-adc-100m14b4cha/wikis/home

[20] *Adc-Lib* project page, https://ohwr.org/project/adc-lib/wikis/home

[21] *SVEC* project page, https://ohwr.org/project/svec/wikis/home

[22] *Fmc-Tdc-1ns-5cha* project page, https://ohwr.org/project/fmc-tdc-1ns-5cha-gw/wikis/home

[23] *Fmc-Del-1ns-4cha* project page, https://ohwr.org/project/fmc-delay-1ns-8cha/wikis/home

[24] IVI Foundation, http://www.ivifoundation.org

[25] *IVI-3.2: Inherent Capabilities Specification*, IVI Foundation, revision 2.1, 9 March 2015.

[26] *IVI-3.15: IviLxiSync Specification*, IVI Foundation, revision 2.0, 9 October 2014.