WRXI: preliminary concept

Dimitris Lampridis

BE-CO-HT

CERN, 3 March 2016
Outline

Introduction
<table>
<thead>
<tr>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Requirements</td>
</tr>
</tbody>
</table>
Outline

Introduction

Requirements

Proposed Approach
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
White Rabbit eXtensions for Instrumentation

- A communication protocol for distributed instrumentation over a White Rabbit (WR) network
- Leverages the high accuracy and precise synchronisation offered by WR
- Augments WR with complex event scheduling, timestamping and real-time message exchanging across the network
- Designed in an application-agnostic way, so that it can be adopted and re-used by others
- Fully open design and implementation
White Rabbit eXtensions for Instrumentation

- A communication protocol for distributed instrumentation over a White Rabbit (WR) network
- Leverages the high accuracy and precise synchronisation offered by WR
- Augments WR with complex event scheduling, timestamping and real-time message exchanging across the network
- Designed in an application-agnostic way, so that it can be adopted and re-used by others
- Fully open design and implementation

Topics for Discussion

- Topics for discussion will be presented like this (and summarised at the end of the presentation)
Purpose

Provide a communication protocol for distributed instrumentation over WR which:

1. Caters to the needs of our CERN users (OASIS in particular)
2. Is generic enough to be re-used in many other distributed instrumentation setups, outside of CERN
Motivation

**OASIS:**
- Open Analog Signals Information System
- Clean, layered software architecture
- Very good “scope” GUI

But...
- Based on ageing, hard to source hardware
- Static, hard-wired
- Limited connectivity
- Problematic scalability
- Limited distances for trigger signals

**WRXI:** preliminary concept
Motivation

Nearest existing commercial solution is **LXI**
- Well-defined standard, maintained by LXI consortium
- Designed for instrumentation
- Works over Ethernet
- Plug & Play
- Has extensions for synchronisation, timestamping and message exchanging
- Solutions for bridging to other networks (GPIB, PXI, …)

But…
- Synchronisation based on IEEE-1588 PTP, not WR
- Some features could be an overkill (eg. embedded webserver per instrument)
- Same transport medium for messages and data could be a problem in complex/busy networks
Vision

Design a new protocol for instrumentation

- Flexible
- Robust
- Scalable
- Re-usable
- Sustainable
- Fully open
- On top of WR
- Compatible with OASIS software architecture

The network will be built on top of WR switches, with distributed instrumentation nodes, under the supervision and control of a host controller. The host controller can be linked to an external network (e.g. CERN technical network). Non-WRXI instrumentation can be attached to special nodes (e.g. GPIB bridges, external trigger generators, etc.)
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
Application-agnostic

Design the WRXI protocol in an application-agnostic way, so that it can be adopted by and re-used in:

- individual setups
- academic organisations
- government organisations
- research institutes
- industrial applications
- commercial products
Layered

- Properly defined layers with clear interfaces
- Allow adoption of the protocol into larger setups
- Maximise re-use of existing infrastructure, tools and services
Layered

- Properly defined layers with clear interfaces
- Allow adoption of the protocol into larger setups
- Maximise re-use of existing infrastructure, tools and services

Example

Connect a “middle” layer of the WRXI network to the OASIS application server, clients and re-use the GUI, while still providing the full layer stack for other independent setups.
Layered

- Properly defined layers with clear interfaces
- Allow adoption of the protocol into larger setups
- Maximise re-use of existing infrastructure, tools and services

Example

Connect a “middle” layer of the WRXI network to the OASIS application server, clients and re-use the GUI, while still providing the full layer stack for other independent setups.

Topics for Discussion

- How/where to connect to the existing OASIS software architecture?
Plug & Play

“zeroconf”-like, newly added nodes can:

- dynamically join a network
- obtain an IP address
- announce their presence
- advertise or convey their capabilities upon request

but still,

- allow the user to assign a “human-friendly” name and description of its function
Plug & Play

“zeroconf”-like, newly added nodes can:

- dynamically join a network
- obtain an IP address
- announce their presence
- advertise or convey their capabilities upon request

but still,

- allow the user to assign a “human-friendly” name and description of its function

Topics for Discussion

- Should nodes be able to learn about the presence and capabilities of other nodes?
- What about node disconnection?
Robust

- Support redundancy (of nodes and/or messages)
- Provide diagnostics (from nodes and switches)
- Distributed event logs (timestamped with WR time)
Synchronisation and Timestamping

- Inherit WR time
- Allow distributed time-based triggering
- Timestamp all outgoing messages using WR time
- Timestamp again all incoming messages upon arrival using WR time
- Timestamp all event log entries
Real-Time Message Exchanging

- Allow exchanging of messages (events, triggers, ...) between nodes
- Host controller estimates message delivery latency based on a priori knowledge of the network topology (information retrieved from WR switches, PTP cores on the nodes, ...)
- Take advantage of the two levels of priority on WR switches to separate between high- and low-priority messages
Real-Time Message Exchanging

- Allow exchanging of messages (events, triggers, ...) between nodes
- Host controller estimates message delivery latency based on a priori knowledge of the network topology (information retrieved from WR switches, PTP cores on the nodes, ...)
- Take advantage of the two levels of priority on WR switches to separate between high- and low-priority messages

**Topics for Discussion**

- How to estimate latency?
- Do we need real-time delivery of measurement data to the host controller?
- Do we use the same link for events and data?
- Do we need TDM (aka. “time slots”)?
Legacy Instrumentation Support

► Provide support for non-WRXI instrumentation, via “bridging” nodes (eg. WR-enabled FMC-DEL receiving WRXI trigger messages and converting them to TTL pulses to be used as external triggers)

► Provide compatibility to existing similar protocols (eg. LXI, more on this later)
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

Example
1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
3. Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
4. Program FMC-DEL to listen for message #2 and generate a delayed TTL pulse upon reception
5. Execute, use delayed TTL pulse from FMC-DEL to trigger a non-WRXI digitiser
6. Retrieve and visualise measurements from FMC-ADC and non-WRXI digitiser
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

Example

- Program FMC-TDC to generate message #1 upon reception of external TTL pulse
- Program FMC-ADC to listen for message #1 and arm upon reception
- Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
- Program FMC-DEL to listen for message #2 and generate a delayed TTL pulse upon reception
- Execute, use delayed TTL pulse from FMC-DEL to trigger a non-WRXI digitiser
- Retrieve and visualise measurements from FMC-ADC and non-WRXI digitiser
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

Example

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

Example

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

**Example**

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
3. Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

**Example**

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
3. Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
4. Program FMC-DEL to listen for message #2 and generate a delayed TTL pulse upon reception
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

**Example**

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
3. Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
4. Program FMC-DEL to listen for message #2 and generate a delayed TTL pulse upon reception
5. Execute, use delayed TTL pulse from FMC-DEL to trigger a non-WRXI digitiser
Remote Control and Sequencing

- Expose all node controls to the host controller
- Allow the host controller to program sequences to the nodes

Example

1. Program FMC-TDC to generate message #1 upon reception of external TTL pulse
2. Program FMC-ADC to listen for message #1 and arm upon reception
3. Program FMC-ADC to generate message #2 upon internal trigger (threshold cross after arm)
4. Program FMC-DEL to listen for message #2 and generate a delayed TTL pulse upon reception
5. Execute, use delayed TTL pulse from FMC-DEL to trigger a non-WRXI digitiser
6. Retrieve and visualise measurements from FMC-ADC and non-WRXI digitiser
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
Overview

Two approaches discussed:

1. Protocol based on LXI
2. Design from scratch
Option #1: Protocol based on LXI

- LAN eXtensions for Instrumentation
- Standard developed and maintained by the LXI Consortium
  - **Strategic**: Keysight, Pickering, Rohde & Schwartz, Rigol
  - **Participating**: Keithley, NI, Tektronix, MathWorks, VTI, ...
  - **Informational**: BK, Spectrum, Teledyne LeCroy, TTi, ...
- Defines a standardised communication protocol over Ethernet
- Aimed at instrumentation, test and measurement systems:
  - scopes, signal/function generators, power supplies, multimeters, logic/network/spectrum analysers, ...

http://www.lxistandard.org
### Option #1: Protocol based on LXI

The LXI specifications include:

- a set of core features for all LXI-enabled instruments
- a set of optional extended features. Of particular interest to WRXI are:
  - Clock Synchronisation
  - Timestamped Data
  - Event Messaging
- All LXI-enabled instruments provide device autodetection
- LXI requires that each device is accompanied by an IVI driver
- IVI “instrument classes” define standard interfaces for most instrumentation devices
- “IVI-3.15: IviLxiSync” defines the API for controlling the arming, triggering, and event functionality of LXI devices

[http://www.ivifoundation.org](http://www.ivifoundation.org)
Option #1: Protocol based on LXI

Clock Synchronisation

- LXI achieves clock synchronisation across an Ethernet network via IEEE-1588 PTP (1588-2008 or later)
- The consortium has defined an LXI IEEE 1588 Profile
- This timebase may be used for a variety of functions including:
  - Timestamping data to expedite post acquisition analysis and ordering
  - Generating LXI Events for precise triggering and synchronisation within an LXI Device or system-wide
  - Generating LXI Event Logs to allow total ordering of LXI Events occurring in all parts of a system
  - Generating synchronous signals in multiple LXI Devices (eg. time-based triggering)
Option #1: Protocol based on LXI

**Timestamped Data**

- LXI devices can assign a timestamp to all measurement data
- When combined with the clock synchronisation extended feature, all such timestamps are derived from the local IEEE 1588 synchronised clock.
<table>
<thead>
<tr>
<th>Option #1: Protocol based on LXI</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Event Messaging</strong></td>
</tr>
<tr>
<td>▶ LXI allows event messaging between nodes within the network, without the need for an external controller</td>
</tr>
<tr>
<td>▶ Multicast UDP (recommended) and unicast TCP support required</td>
</tr>
<tr>
<td>▶ “Events” may include:</td>
</tr>
<tr>
<td>▶ LXI Events specified in this standard</td>
</tr>
<tr>
<td>▶ LXI Device-specific events specified by the vendor</td>
</tr>
<tr>
<td>▶ Application-specific events specified by the user</td>
</tr>
<tr>
<td>▶ Optional message response for reliable communication over UDP</td>
</tr>
</tbody>
</table>
Option #1: Protocol based on LXI

**LXI Event Message Format**

<table>
<thead>
<tr>
<th>HW Detect</th>
<th>Domain</th>
<th>Event ID</th>
<th>Sequence</th>
<th>Timestamp</th>
<th>Epoch</th>
<th>Flags</th>
<th>Data Field(s)</th>
<th>Zero</th>
</tr>
</thead>
</table>

**HW Detect**: char[3], magic value, defaults to “LXI”

**Domain**: uint8, separation of network into virtual domains

**Event ID**: char[16], event identifier

**Sequence**: uint32, message counter

**Timestamp**: uint32 seconds, uint32 nanoseconds, uint16 fractional_nano

**Epoch**: uint16, 16 MSB of the IEEE 1588 seconds field

**Flags**: uint16, various protocol flags

**Data Field(s)**: each data field includes:

- uint16, length of payload data
- uint8, type of data to follow. Standard defines values for most common types (uint, ASCII, XML, JSON, …)
- any number of bytes

**Zero**: two bytes with zero value
Option #1: Protocol based on LXI

Three possible scenarios for an LXI-based WRXI

1. **Custom LXI-based protocol.** Borrowing (heavily) from LXI. Requirements covered by LXI if possible, custom solutions for everything else. Unnecessary LXI features ignored.

2. **Partial LXI compatibility.** Requirements are covered by LXI wherever possible, even if this means reduced functionality. Unnecessary LXI features ignored if they don’t break interoperability between LXI and WRXI.

3. **Full LXI conformance.** Any additional functionality may only be implemented using “vendor-specific” fields, etc. No LXI features ignored (e.g. TCP listening sockets supporting up to 8 connections, embedded web-servers per instrument).
Option #2: Design from Scratch

- Maximum risk, maximum effort, maximum flexibility
- Makes sense if a “custom LXI-based protocol” cannot meet our design goals/specifications (e.g. lack of front-end computer might make it very difficult to implement many of LXI’s features)
- In this case, we should also consider using the WR node core project (aka. Mock Turtle) in a way similar to the one used in LIST and D3S
  - Flexible and easy to develop
  - Does not require a front-end computer on the node
  - Lack of block memory for smaller FPGAs (e.g. SPEC)
  - Remote message queues connected via Wishbone (over Etherbone)
Option #2: Design from Scratch

- Maximum risk, maximum effort, maximum flexibility
- Makes sense if a “custom LXI-based protocol” cannot meet our design goals/specifications (e.g., lack of front-end computer might make it very difficult to implement many of LXI’s features)
- In this case, we should also consider using the WR node core project (aka. Mock Turtle) in a way similar to the one used in LIST and D3S
  - Flexible and easy to develop
  - Does not require a front-end computer on the node
  - Lack of block memory for smaller FPGAs (e.g., SPEC)
  - Remote message queues connected via Wishbone (over Etherbone)

Topics for Discussion

- Can we safely assume that WRXI nodes will have a front-end computer?
- WRXI protocol: which approach?
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
Overview

WRXI: preliminary concept
Clock Domains

- 125 MHz system clock derived from 20 MHz VCXO
- All 125 MHz clocks from CDCM61004 unused
- Both VC DACs unused
- FMC-ADC has its own Si570 I2C-programmable oscillator for the sampling clock
- ADC sampling clock upper limit is 100 MHz
- “ADC core” crosses clock domains
- “Timetag core” is on system clock domain
Options

1. Make use of WR timing interface
   - Use the 125 MHz FPGA clock from the CDCM61004 both as system clock and reference clock
   - Derive a 62.5 MHz clock from the system clock and use it as local system clock for WR-PTP core
   - Derive a 62.5 MHz clock from the 20 MHz VCXO and use it as DMTD clock for WR-PTP core
   - Let WR-PTP core control the two VC DACs of the carrier
   - Upgrade “timetag core” to get timestamps from timing interface of WR-PTP core

2. Discipline the ADC sampling clock
   - Feed the ADC sampling clock as an auxiliary clock to the WR-PTP core
   - re-use/update existing wr_Si57x_interface to close the control loop
Potential Issues

- Issues with using the WR timing interface:
  - WR-PTP core wishbone interface on different clock domain (div by 2, no phase shift wrt to FPGA clock). Will need a clock crossing module.

- Issues with feeding the ADC sampling clock to WR-PTP core:
  - 100 MHz sampling clock frequency (still OK since \( \text{LCM}(10 \text{ ns}, 8 \text{ ns}) = 40 \text{ ns} \), can be aligned every five 125 MHz ticks)
  - Need to investigate if we can phase shift the Si570 fast enough
## Potential Issues

- **Issues with using the WR timing interface:**
  - WR-PTP core wishbone interface on different clock domain (div by 2, no phase shift wrt to FPGA clock). Will need a clock crossing module.

- **Issues with feeding the ADC sampling clock to WR-PTP core:**
  - 100 MHz sampling clock frequency (still OK since \( \text{LCM}(10 \text{ ns}, 8 \text{ ns}) = 40 \text{ ns}, \text{can be aligned every five } 125 \text{ MHz ticks} )
  - Need to investigate if we can phase shift the Si570 fast enough

## Topics for Discussion

- Do we need to discipline the ADC sampling clock?
- What about other digitisers with different sampling frequencies and/or free-running clocks that we cannot control?
Proposed Approach

WRXI: preliminary concept
Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion

Proposed Approach

WRXI: preliminary concept
Outline

Introduction

Requirements

Proposed Approach

WR-enable the FMC-ADC

Topics for Discussion
## Topics for Discussion

### WRXI protocol
- WRXI protocol: which approach?
- Can we safely assume that WRXI nodes will have a front-end computer?
- Should nodes be able to learn about the presence and capabilities of other nodes?
- What about node disconnection?
- How to estimate latency?
- Do we need real-time delivery of measurement data to the host controller?
- Do we use the same link for events and data?
- Do we need TDM (aka. “time slots”)?
- How/where to connect to the existing OASIS software architecture?

### FMC-ADC
- Do we need to discipline the ADC sampling clock?
- What about other digitisers with different sampling frequencies and/or free-running clocks that we cannot control?