## Contents

1 Introduction ........................................... 3  
1.1 About the White Rabbit technology .................... 3  
1.2 About the Starting Kit ................................. 3  
1.3 About this document: ................................... 4  

2 System Setup ......................................... 4  
2.1 Starting kit components ................................ 4  
2.2 Physical configurations .................................. 5  

3 Installation WR-NIC .................................... 5  
3.1 Tools ....................................................... 8  
3.2 Install the project ....................................... 8  
3.3 Structure of the spec-sw project ....................... 8  
3.4 Compile & install ....................................... 9  
3.5 Download, install & load the gateware .................. 9  
3.6 Loading the driver .................................... 10  

4 Setting Master & Slave using White Rabbit Core ........ 10  
4.1 Connect to the UARTs ................................... 10  
4.2 The White Rabbit Core Shell ............................ 11  
4.3 Configure in slave & master mode ...................... 12  
4.4 Bring up the network interface ........................... 13  

5 The Experiments ....................................... 14  
5.1 Playing with the DIO channels ......................... 14  
5.2 Synchronization of PPS .................................. 17  
5.3 The Network Card Interface (NIC) Synchronization ... 18  
5.4 Transmitting external frequency ........................ 20  

6 Quick Start Guide For Developers ....................... 24  
6.1 Introduction ............................................. 24  
6.2 Tools ...................................................... 24  
6.3 Golden SPEC gateware .................................. 25  
6.4 WRP-SW (LM32 firmware) ............................... 25  
6.5 WR-NIC (HDL-gateware) ............................... 26
1 Introduction

1.1 About the White Rabbit technology

White Rabbit is an extension to Ethernet network with accurate synchronization and Gigabit data transfers. It has been conceived to fulfill the following goal.

**Time Precision** Provides a common clock for physical layer in the entire network, allowing a sub-nanosecond accuracy and picoseconds precision of synchronization.

**Scalability** The WR network is designed to be highly scalable with up to thousands of nodes. It also intents to be as modular as possible and compatible with non-WR devices. Finally it is fully open hardware, firmware and software so you can develop your own application.

**Distance Range** In order to handle the majority of industrial and scientific facilities, the WR network is designed to support distance ranges of 10 km between nodes using fiber cables.

The Starting Kit has been designed to test and check if the White Rabbit technology meets your requirements and also to ease the understanding on what can do with it and how to integrate it on your own project.

You can find more information on other components and use case on:


1.2 About the Starting Kit

This starting kit uses two SPEC + FMC-DIO cards also called nodes in the White Rabbit world. Each node makes basic operations such as input timestamping or programmable output pulse generation. Additionally, specific software and gateware layers allow to use it as a standard network interface card implementing the White Rabbit technology functionalities. Network packages with accurate time-stamping information are generated/timestamped at the hardware level to achieve the highest accuracy.

It is based on different project:

- **spec-sw**: driver to communicate to the card through PCIe and a set of tools to experiments
- **wr-nic**: gateware that includes the NIC & DIO capabilities.
- **wrpc-sw**: white rabbit PTP firmware for the synchronization.
1.3 About this document:

This document is intended as a step by step user friendly tutorial to start in the White Rabbit “World” with simple experiments. Some concepts are deliberately avoided to ease the comprehension of this document. If you want to know more about these concepts you should look at the documents in the Reference section and especially to the following pdfs:

- spec-sw.pdf
- wrpc.pdf

The rest of the document provides an explanation about the system setup, software driver, FPGA configuration and application examples utilization.

This will allow to easily validate the main capabilities of White Rabbit technology and works as first stage for getting familiar with this.

There is also a chapter called Quick Start Guide for Developers in order to make it easy for them to find the different source project and how to compile them.

2 System Setup

This section describes the main elements required to start with the White-Rabbit based on the “White Rabbit (SPEC-based) starting kit”. The different hardware elements and its connection are described in this section.

2.1 Starting kit components

The starting kit is composed of various elements that you should find in the box\(^1\):

- 2x SPECs boards compatible with PCIe x4 or more
- 2x FMC DIOs 5CH TTL A
- 2x SFPs LC
  - AXGE-1254-0531 (blue)
  - AXGE-3454-0531 (violet)
- 1x LC-LC cable (2m)
- 2x LEMO cable (2m)
- 2x LEMO-BNC Adaptor

\(^1\)Only with the standard version of starting kit.
2.2 Physical configurations

- Plug the SPEC+DIO into your PCIe x16 port.
- You should also connect a micro-USB cable to the UART of the board. (A Virtual UART is also available, but is safer to use the physical one)
- Connect the two SPEC+DIO boards using the SFPs and the optical fiber cable (LC-LC)
- Run a linux distribution (This tutorial has been written using Ubuntu 12.04 and it is focused on debian’s like distribution).
- Prepare an oscilloscope with at least two channels input near the DIO output.

Note: This tutorial follow a configuration with two SPEC+DIO boards connected in the two PCIe x16 interfaces of the same computer.
You can also use two different computers (if you do not have two PCIe x4) and follow this tutorial; you just need to replace all the commands with wr1 or 0x0300 by the wr0 and its corresponding bus_id on the second PC.

The configuration with two separated PCs is more “natural” to understand how communicate the starting kit, but it requires more space to realize it.

3 Installation WR-NIC

Once you have working linux with the boards plugged:

- Install the tools to builds drivers, etc...
- The drivers: spec-sw
- The HDL bitstream of SPEC+DIO: wr-nic
Figure 2: Configuration with one or two PCs

Figure 3: Mounting two SPECs in the same PC
Figure 4: Connecting SFPs and fiber

Figure 5: The UART use a micro-USB cable to communicate
3.1 Tools

The first step is to install all the tools that you will need:

- **git**: A powerful & distributed version control system
- **build-essentials**: Contains various binaries to build source code
- **linux-source**: Might be useful to compile drivers & kernel modules
- **minicom**: Hyperterminal for linux
- **texinfo, texlive, emacs**: Tools to build documentation

You can also run this command:

```
sudo apt-get install git build-essential linux-source minicom texinfo emacs texlive
```

3.2 Install the project

The first step is to install the driver to communicate with the card using the PCIe port. To do so you need to get the latest release of **wr-starting-kit** project which includes the **spec-sw** project.

```bash
## Create the "root" directory
>:$ mkdir -p ~/wr/
>:$ cd ~/wr/

## Clone the repository
>:$ git clone git://ohwr.org/white-rabbit/wr-starting-kit.git
>:$ cd wr-starting-kit

## Checkout the stable release
>:$ git checkout -b wr-starting-kit-v1.0 wr-starting-kit-v1.0

## Obtain the spec-sw project using submodules
>:$ git submodule init
>:$ git submodule update
```

3.3 Structure of the spec-sw project

Most of the information on how the project is structured can be find in the **spec-sw.pdf** in the /doc folder, however we are going to briefly resume it:

3.3.1 Folders layout

Each of the following folder contains:

- **doc**: documentation of the project

---

2You should only use our package or proper release (tagged commit). The **master** branch might have the latest source but support is only offered for tagged release. The other branches are normally used for development and are not stable.
3.3.2 Structure of the driver and HDL

The starting kit is contains 3 different components:

- **fmc.ko**: Define a generic FMC-bus used to identify and connect to FMC boards independently of the carrier being used (e.g. SPEC, SVEC, ...)
- **spec.ko**: It is the device driver that connect to the SPEC through PCI. It also loads the spec-init.bin “golden” gateware file.
- **wr-nic.ko**: The wr-nic driver is basically an Ethernet driver with support for hardware time stamping. It also loads the
  - `fmc/wr_nic_dio.bin` gateware for the NIC and DIO properties.
  - `fmc/wr_nic_dio-wrc.bin` software for the LM32.

3.4 Compile & install

A Makefile in the `spec-sw` project has been written to compile and install easily the drivers and the tools used below.

```bash
## Go to spec-sw project
>:$ cd spec-sw

## In the root folder (spec-sw), run
>:$ make

## then, install the driver in your system so that they load automatically
>:$ sudo make install
```

If everything works well you should see the driver in

```bash
>:$ ls -l /lib/modules/$(uname -r)/extra
```

**Notes:** In some distribution such as Ubuntu 12.04, you might need to force the creation of dependancy to load the modules using `modprobe`.

```bash
>:$ sudo depmod -a
```

3.5 Download, install & load the gateware

You need to install the gateware to `/lib/firmware/fmc`

---

3 More information about the fmc-bus can be found within `fmc-bus.pdf` in the `doc/` folder.
4 In future version, the program file for the LM32 will be included into the gateware.
5 The HDL binaries (gateware) came from other project, you can find in the Developers Section where to obtain them and how to compile them.
3.5.1 Manual installation

```bash
## Create the gateware folder
>: $ mkdir firmware
>: $ cd firmware

## Download all the firmware files (from OHWR website).
>: $ wget -nv http://www.ohwr.org/attachments/download/1815/wr-starting-kit-v1.0_gw.tar.gz
   -O- | tar -xvz
```

Once you have the file you need to install them to your system in order to make them load automatically.

```bash
## Install the HDL binaries in /lib/firmware/fmc (require sudo)
>: $ sudo cp -v firmware/*.bin /lib/firmware/fmc
```

3.5.2 Automatic installation

You can also use the script `wr-ssk-get` in `wr-starting-kit/scripts` folder to ease the installation. `sudo` is required.

```bash
## Fetch and install the firmware
>: $ scripts/wr-ssk-get --all
```

3.6 Loading the driver

To enable the wr-nic you should execute the `modprobe` command to load everything.

```bash
>: $ sudo modprobe wr-nic
```

You should expect to obtain one (or two) new interface(s):

```bash
>: $ ifconfig -a | grep wr
wr0 Link encap:Ethernet HWaddr XX:XX:XX:XX:XX:XX
```

4 Setting Master & Slave using White Rabbit Core

A “SPEC+FMC DIO” board is considered as a slave node, however in a standard White Rabbit network the synchronization is done using a master such as the White Rabbit switch. In order to enable the synchronization capabilities we need to set up one board in master mode (faking to be the switch) and the other one in slave mode (default mode). This is done through the shell of White Rabbit Core that give this functionality among others.

4.1 Connect to the UARTs

4.1.1 Physical UART

To configure the SPEC board in master/slave mode you need to access to the White Rabbit Core (WRC) through the UART (virtual or physical).

---

6 We have found a problem in some distribution with `modprobe` command. You should use `insmod` instead and look at the Know bugs sections.
To access to the physical, you first need to connect the mini-USB of the SPEC to your PC. Then, you need to run a terminal emulator (such as minicom) on the device created while plug-in the USB cable.

```
>:$ sudo minicom --device=/dev/ttyUSB0 -b 115200
>:$ sudo minicom --device=/dev/ttyUSB1 -b 115200
```

**Notes:** *ttyUSB0 might not correspond to your first board, and ttyUSB1 to the second one. This depend on how you plug the USB cable to your USB of your machine.*

### 4.1.2 Virtual UART

An easier way to perform this operation is to access to the virtual uart of the SPEC board by using the spec-vuart tool. You first need to know which is the bus_id of your board.

**Notes:** *The VUART is currently not working for this package, but the information is given for the next version. See Know bugs sections*

```
>:$ lspci | grep CERN
02:00.0 Non-VGA unclassified device: CERN/ECP/EDU Device 018d (rev 03)
03:00.0 Non-VGA unclassified device: CERN/ECP/EDU Device 018d (rev 03)
```

This mean that your first board (wr0) is at 02:00.0 and the second one (wr1) is at 03:00.0 Thus, for each board you can open a terminal and run the following command:

```
>:$ sudo tools/spec-vuart -b 02 #For the first SPEC board
>:$ sudo tools/spec-vuart -b 03 #For the second SPEC board
```

### 4.2 The White Rabbit Core Shell

Once you are in the UART you can should obtain the White Rabbit Core console (*wrc#*). A complete reference of the shell commands is included in the *wrpc.pdf* manual or you can read them in the Wiki.

The most useful commands are repeated here for your convenience

- `gui`
- `mode master`
- `mode slave`
- `ptp stop`
- `ptp start`

For the tutorial we will name:

- `wrc1#` for *wrc#* console of the main board (wr0/busID=0x0200)
- `wrc2#` for *wrc#* console of the second board (wr1/busID=0x0300)

---

7On Ubuntu 12.04, you need sudo permission to access to a ttyUSBX device. It might be a good idea to add your user to the dialout group to obtain appropriate permission on the devices: *sudo usermod -a -G dialout $USER*
4.3 Configure in slave & master mode

Thus know we can perform the following operation:

- Check the if you have the correct SFP and its corresponding value:

```
wrcl# sfp detect
AXGE-3454-0531 #purple
wrcl# sfp match
SFP matched, dT=46407, dR=167843, alpha=-73622176
```

```
wrcl# sfp detect
AXGE-1254-0531 #blue
wrcl# sfp match
SFP matched, dT=46407, dR=167843, alpha=73622176
```

- Setup one board in master (the one with the purple SFP)

```
wrcl# mode master
SPLL_Init: running as Free-running Master, 1 ref channels, 2 out channels
Locking PLL...
```

- And let the other one in slave mode (this should not be necessecary but you might have set it before in master mode)

```
wrcl# mode slave
slave
```

4.3.1 Start the synchronization with PTP deamon

You should run the PTP daemon on both card

```
wrcl# ptp start
wrcl# ptp start
```

And you should obtain the following message on the slave board:

```
SPLL_Init: running as Slave, 1 ref channels, 2 out channels
Enabling ptracker channel: 0
Enabling ptracker channel: 0
servo:pre: -1885143040/2ps
servo:Deltas: Master: Tx=46407ps, Rx=175043ps, Slave: tx=46407ps, Rx=167843ps.
Adjust: counter = seconds [+0]
Adjust: counter = nanoseconds [+343914800]
servo:busy
```

Finally you can run the WRC shell in gui mode to obtain more information

```
wrcl# gui
```

8If you obtain an error such as **Could not match to DB** you might add your own SFP parameters. List your actual sfp database with `sfp show` and check the `wrpc.pdf` that explain how to add SFPs parameters.
4 Setting Master & Slave using White Rabbit Core

WR PTP Core Sync Monitor v 1.0
Esc = exit

TAI Time: Thu, Jan 1, 1970, 0:30:15

wru1: Link up (RX: 766, TX: 340), mode: WR Slave Locked Calibrated

Synchronization status:
Servo state: TRACK_PHASE
Phase tracking: ON
Synchronization source: wru1
Aux clock status:

Timing parameters:
Round-trip time (mu): 698557 ps
Master-slave delay: 345660 ps
Master PHY delays: TX: 46407 ps, RX: 175043 ps
Slave PHY delays: TX: 46407 ps, RX: 167843 ps
Total link asymmetry: 7237 ps
Cable rtt delay: 262857 ps
Clock offset: 2 ps
Phase setpoint: 7268 ps
Skew: 2 ps
Manual phase adjustment: 0 ps
Update counter: 117

You can see that the slave node is locked, calibrated and that the phase tracking is enabled.

*Notes: We also recommend you to set up an init script if you do not want to repeat these operations at each reboot. You can look at the wrpc.pdf for more information or use the init show command to check the one you have running.

4.4 Bring up the network interface

Once the drivers are loaded and the PTP has started you might need to bring the new network interfaces up. In order to do this you should just execute:

>:$ sudo ifconfig wr0
>:$ sudo ifconfig wr1

And then check if they are mounted by doing

>:$ sudo ifconfig | grep wr
wr0 Link encap:Ethernet HWaddr 08:00:30:0d:e8:6b
wr1 Link encap:Ethernet HWaddr 08:00:30:0d:e4:cd

*Notes: On distribution that use gnome desktop (i.e, Ubuntu), the interfaces are automatically mounted by gnome network manager, so you might not need to perform the last step.
5 The Experiments

To perform the experiments we recommend you to plug the LEMO cable to your oscilloscope by using the LEMO-BNC adapter provided in your package.

It is also recommended to follow the order of the tutorial as the concept you need to understand are explained in the same order as the experiments.

Notes: Before each experiment you should be sure to set up that your boards are in master/slave mode and that the PTP deamon are running on both.

5.1 Playing with the DIO channels

The first step to perform is to check if the DIO channels are accessible using the installed driver. In the tools subdirectory you will you find the wr-dio-cmd program, that let you quickly test the I/O features of the DIO boards. This is the general syntax of the command: wr-dio-cmd <ifname> <cmd> [arg] ...

The main commands correspond to: mode, pulse & stamp, and you can find an enhanced description of them in the spec-sw.pdf document.

5.1.1 Setting the mode

The first thing is to set the correct mode for each channel, you can use a command that set the mode for a specific channel or for all of them.

```
mode <channel> <mode> [<channel> <mode> ...]
mode <modech0><modech1><modech2><modech3><modech4>
```

The available modes are:

- **I/i**: input (I with 50-ohm termination resistor and i without)
- **0/1**: output, steady state fixed at 0 low or 1 high.
- **D/d**: DIO core output (D with 50-ohm termination resistor and d without)

For example you can set it up like this

```
$ sudo wr-dio-cmd wr0 mode Ii--0
```

Notes: please note that the pulse command activate the DIO mode (without changing the termination)

5.1.2 Generating pulse

The command is simply described as:

```
pulse <channel> <duration> <when> [<period> <count>]
```
5 The Experiments

Figure 6: Connecting LEMO on connector #5 (channel 4)

You should plug the LEMO cable in the connector #5 (channel 4), and connect the BNC adapter to your oscilloscope.

Finally you need to trigger pulse in your oscilloscope. Then you can try the following commands:

```
## Pulse channel 4 for 0.1 seconds now
>:$ sudo wr-dio-cmd wr0 pulse 4 .1 now

## Pulse for 10 microseconds in the middle of the next second
>:$ sudo wr-dio-cmd wr0 pulse 4 .00001 +1.5

## Pulse for 1ms at 17:00 today
## Set the datetime of the next event (60 seconds from now)
## NOTE: this only work if the date is correctly set in the master SPEC,
>:$ sudo wr-dio-cmd wr0 pulse 4 .001 $(date +%s --date 17:00)
```

5.1.3 Time stamping output

Once you have generated the pulse you can retrieve its timestamp by executing:

```
>:$ sudo wr-dio-cmd wr0 stamp 4
ch 4, 378.78858536
ch 4, 381.268701864
ch 4, 387.284885816
ch 4, 469.500000000
ch 4, 500.500000000
ch 4, 504.500000000
ch 4, 530.500000000
ch 4, 534.500000000
ch 4, 542.500000000
```

9 As the application use ioctl you should call with root privilege by using sudo.
10 The connector on the FMC DIO panel are enumerated from 1 to 5 and correspond to the channel 0 to 4 on the wr-dio-cmd.
5.1.4 Time stamping input & output.

The idea here, is to create a pulse from one DIO and to timestamp it as local and remote input. Then we should compare the both time lapse between generation and reception.

The configuration is done as in the figure below:

![Figure 7: Time-stamping configuration](image)

```bash
## Configure channel 2 of both boards (wr0 & wr1) as input with termination
>:$ sudo wr-dio-cmd wr0 mode 2 I
>:$ sudo wr-dio-cmd wr1 mode 2 I

## Then flush the previous timestamp
>:$ sudo wr-dio-cmd wr0 stamp &> /dev/null
>:$ sudo wr-dio-cmd wr1 stamp &> /dev/null

## Schedule the pulse to the next common event on two outputs
>:$ sudo wr-dio-cmd wr0 pulse 0 .00001 +2
>:$ sudo wr-dio-cmd wr0 pulse 4 .00001 +2

## Then (after 60s), you should run stamp on the wr0
>:$ sudo wr-dio-cmd wr0 stamp
ch 0, 2267.50000000
ch 2, 2267.50000008
ch 4, 2270.50000000
>:$ sudo wr-dio-cmd wr1 stamp
ch 2, 2270.50000008
```

We observe that locally (output=>ch0, input=>ch2) and remotely (output=>ch4, input=>ch2), the timestamp between sending a pulse and receiving is of 8 nanoseconds\(^{11}\) for both. This is a very simple example to show how the two SPECs are synchronized.

\(^{11}\) The DIO core work at a frequency of 125Mhz, therefore the minimum difference between two timestamped events is of 8ns. However with the next experiment you will find out that the synchronization is better.
5.2 Synchronization of PPS

This correspond to the experiment from spec-sw.pdf:6.8 document. Here we want to synchronize the PPS of the master and slave board using WR protocol on a fiber.

5.2.1 Setup

To obtain more “impressive” result, this experiment should be realize with two SPEC+DIO boards separated by a 5km fiber on two different PCs. However to simplify the experiments we are using 2m fiber cable and the two boards are connected to the same PC. One channel of each board (channel 4 & 0) is connected to the oscilloscope using LEMO cable & adapter.

![Figure 8: Setup for the PPS synchronization](image)

5.2.2 Run PPS command

Once the synchronization is enabled, you just need to run the PPS from the command line. This program just fires a 1ms-long pps pulse on one of the output channels.

```
## run pps on channel 2 of the default SPEC card
>:$ sudo tools/wr-dio-pps wr0 4

## run pps on channel 0 of the "second" card
>:$ sudo tools/wr-dio-pps wr1 0
```

5.2.3 Result

The following two figures show two such pulse-per-second signals retrieved from two different simple-DIO cards.

To avoid the unstable signal after the pps go high you can add a 50-ohm resistor termination if the oscilloscope let you do it.

To stop the PPS you need to set the channel in output mode (low/high)
Figure 9: Synchronization of the two PPS (5ns/div)

```
sudo tools/wr-dio-cmd wr0 mode 4 0
sudo tools/wr-dio-cmd wr1 mode 0 0
```

**Notes:** In some cases, it may happen that the leading edges differ by almost exactly 8ns; this happens because in the simple-DIO all times are quantized by 8ns-long clock ticks. The differences in internal delays (which depend on the carrier, the mezzanine and the FPGA binary), are not self-measured and calibrated in this simple design and may appear in the output after quantization.

5.3 The Network Card Interface (NIC) Synchronization

5.3.1 Introduction

The `wr-nic` driver registers a Linux network interface card for each SPEC device it drives. The cards are called `wr%d` (i.e., `wr0`, `wr1`, ...).

The SPEC can carry normal data traffic in addition to the PTP frames of White Rabbit, that remain invisible to the host computer.

**Notes:** If a user want to use the WR-NIC on a real network to transfer data he should assign an Interface address. However the White Rabbit synchronization happens at Ethernet layer therefore no IP address is need. Moreover if you want to communicate two WR-NIC boards that are on the same computer you should not use an IP protocol because the data will never be put on the fiber as the computer know that the two IPs are on the same machine.

5.3.2 Timestamp Mechanism

The SPEC Ethernet interface supports hardware timestamping for user frames through the standard Linux mechanisms. Time stamps are currently reported with a resolution of 8ns only (White Rabbit does much better, but we don’t have the code in place for this demo, yet).

Unfortunately the Linux mechanisms are not trivial: the application must enable timestamping on both the hardware interface and the specific socket it is using, and it must issue several `ioctl` and `setsockopt` commands. Moreover, timestamps are returned to user space using the `recvmsg` system call, which is more difficult to deal with than the normal `send` or `recv`.

www.sevensols.com/whiterabbitsolution
To simplify use of timestamps for Ethernet frames, this package includes the `stamp-frame` program in the `tools` directory. The program is a minimal implementation of the basic time-synchronization protocols (like NTP and PTP), but excluding the synchronization itself. The idea is sending a frame from one host to another, and receiving a second frame back; the departure and arrival times are recorded and collected at a single place, so they can be reported to the user.

5.3.3 Synchronization And Timestamping

The `stamp-frame` example supports two modes of operations. In `listen` mode, it binds to an Ethernet interface and listens forever: it waits for the forward frames and replies to them; in normal mode it sends the forward frame and reports data as soon as it gets a reply.

```bash
# On a terminal run
>$ sudo tools/stamp-frame wr0 listen
tools/stamp-frame: Using interface wr0, with all timestamp options active

# On another terminal (maybe on another host) run
>$ sudo tools/stamp-frame wr1
tools/stamp-frame: Using interface wr1, with all timestamp options active
timestamp T1: 1891.948736656
timestamp T2: 1892.038390176
timestamp T3: 1892.048625152
timestamp T4: 1891.958972320
round trip time: 0.000000688
forward time: 0.089653520
backward time: -1.910347168
```

The four times are departure and arrival of the forward frame, followed by departure and arrival of the backward frame. Thus, time stamps T1 and T4 are collected at the original sender (here: `wr1`) while T2 and T3 are collected at the remote host (here: `wr0`). The times above are all consistent because the two SPEC cards are synchronized with White Rabbit. The reported forward and backward times match the fact that I used a 10km fiber to connect the two cards; the difference between them is due to the different speed of light in the two directions, because the two SFP transceivers I plugged use different wave lengths.

The following example shows the output for two forcibly-unsynchronized cards. The difference between the two clocks is clearly a few seconds; the round trip time is correct nonetheless, because it is a difference of differences:

```bash
timestamp T1: 13.225249168
timestamp T2: 9.130237600
timestamp T3: 9.140438816
timestamp T4: 13.235559016
round trip time: 0.000108632
forward time: -5.904988432
backward time: 4.095120200
```

The code in `stamp-frame` is designed to be simple to be reused, but there is one non-obvious detail that is worth describing here. Whereas the receive timestamp is returned to user-space together with the frame it refers to, the transmit timestamp is only known after the relevant frame left the computer. For this reason, in order to communicate the TX timestamp of a frame to your peer, you’ll need to send another message which carries the departure time of the previous frame. This further message is usually called follow-up, and `stamp-frame` respects this tradition.
5.4 Transmitting external frequency

This experiment merges the concepts seen before with the DIO channels and the NIC synchronization.

5.4.1 Resume

A typical application for White Rabbit (or any time synchronization system) is being able to generate output signals at the same time in different output boards; another typical application is time stamping input events.

Thus in these experiments we are going to transmit an external frequency in the 100Hz range using the starting kit:

- The user supplies a ~100Hz square wave on the channel 0 of the master card.
- The dio-ruler on master host reads the UTC time of the rising edge of the external pulse upon IRQ.
- Then the dio-ruler forward it (as an ioctl structure) to its local and remote selected outputs.
- At the slave workstation the dio-agent wait to receive timestamped ioctl order and pass it to the DIO driver.
- The DIO will generate a 1ms pulse at the programmed time.
- On the scope we should see a constant time offset between the two pulses.

The figure below illustrate how the connection must be done between the different components:

![Figure 10: Setup to transmit a 100Hz signal](www.sevensols.com/whiterabbitsolution)
5.4.2 How does it work?

The example is made up of two programs: `wr-dio-agent` and `wr-dio-ruler`: the ruler rules and the agents acts only.

These programs transmit using raw Ethernet frames to force the optical fiber transmission when two boards are on the same machine. It also transmit using broadcast address to avoid complication on selecting to who it should transmit. The simplification above, however, most likely prevents the programs from working within a more complex network topology.

**Notes:** By using an IP protocol (i.e., UDP) the data will never enter the optical fiber between the two boards because the OS see that the destination is local and therefore route them directly to the receiver without using the sender.

The agent is a “dumb” agent in charge of forwarding WR packet to the DIO core. The ruler waits for timestamps to appear on a specific input channel; when a positive-going edge occurs the ruler notified it or replicates the edge on one or more outputs. Each output can be local or remote, and can use a different delay from the input pulse.

For further knowledge We strongly recommend you the lecture of the spec-sw.pdf:6.8 document and to also take a look at the source code of `wr-dio-agent` and `wr-dio-ruler`

**Notes:** All pulses generation are driven by host software, after a hardware interrupt reports the input event. For this reason, you’ll not be able to reliably replicate pulses with delays smaller than a few hundred microseconds, depending on the processing power of your computer and the load introduced by other processes. For remote connections, you must also count the overhead of network communication as well as transmission delays over the fiber (a 10km fiber introduces a delay of 50 microseconds).

Then we should run the “dumb” agent[agentbg] on the slave board in charge of forwarding ioctl packet to the DIO core:

```
$ sudo wr-dio-agent wr1 &
```

Then you need to connect the output of the generator to channel 0 of the master SPEC+DIO board. The generated waveform should be a 0-5V pulse at ~100Hz (5ms at 5V then 5ms at 0V)

You should setup the channel 0 as input and check if the 100Hz signal is correctly timestamped:

```
>:$ sudo wr-dio-cmd wr0 mode 0 I
```

```
## Set channel 0 of master board as Input.
>:$ sudo wr-dio-cmd wr0 stamp
```

```
ch 0, 3573.281851462
ch 0, 3573.291851460
ch 0, 3573.301851460
ch 0, 3573.311851456
ch 0, 3573.321851454
ch 0, 3573.331851458
ch 0, 3573.341851454
...
```

Here we can see that I correctly timestamp each 10ms (+/- 10ns)\(^\text{13}\).

\(^\text{12}\)If you lack a wave form generator, you can make an output pulse with `wr-dio-pps` or other means and use it as a trigger. Check the Two PCs example.

\(^\text{13}\)The 10 nanosecon jitter is due to our waveform generators used in the experiments. This should disappear by using a good quality waveform generator.
The Experiments

Once you know that your input is correct, you need to capture each event on channel 0 and replicate them with a delay of 1ms on both the local (channel 3) and the remote card (channel 1)

## Creating the ruler to forward from input 0 to local3 and remote1

```bash
$ sudo wr-dio-ruler wr0 IN0 L3+0.001 R1+0.001
wr-dio-ruler: configured for local channel 3, delay 0.001000000
wr-dio-ruler: configured for remote channel 1, delay 0.001000000
```

5.4.3 Result

To check if the experiment works well you can compare the timestamp of the two outputs

## Checking the timestamp on local output

```bash
$ sudo wr-dio-cmd wr0 stamp 3
ch 3, 5340.004864960
ch 3, 5340.014864960
ch 3, 5340.024864968
ch 3, 5340.034864964
...
```

## Checking the timestamp on remote output

```bash
$ sudo wr-dio-cmd wr1 stamp 1
ch 1, 5340.004864962
ch 1, 5340.014864960
ch 1, 5340.024864968
ch 1, 5340.034864960
...
```

Or compare the signals\(^{14}\) on the scope as in the next figure:

![Figure 11: Result with a 100Hz train](image)

Notes: This code is think as a demo sample therefore: the delays should be no more than the interval between input pulses, because the tools reprograms all output triggers at each input event. The output pulse is fixed at 1ms

---

\(^{14}\)To do this you need to connect from the waveform generator to the input with a LEMO cable that is not provided.
5.4.4 Two PCs example

This experiment is taken from the spec-sw document; we verified the tools and we think this is a good explanation of how to use them.

It shows how to use the ruler and agent on two hosts, called spusa and tornado. It is pretty the same thing as the previous experiment only that the board are both name wr0.

![Two PCs example diagram](image)

Figure 12: Transmit PPS between two PCs

The input events on spusa are replicated to one local channel and two remote channels, with a delay of 1ms. The input events in this case are from a pulse-per-second signal from channel 0.

```plaintext
>spusa:# wr-dio-pps wr0 0
>tornado:# wr-dio-agent wr0 &

>spusa:# wr-dio-ruler wr0 IN4 L3+.001 R4+.001 R2+.001
wr-dio-ruler: configured for local channel 3, delay 0.001000000
wr-dio-ruler: configured for remote channel 4, delay 0.001000000
wr-dio-ruler: configured for remote channel 2, delay 0.001000000

[... wait a few seconds ...]

>spusa:# wr-dio-cmd wr0 stamp 3
ch 3, 385.001000000
ch 3, 386.001000000
ch 3, 387.001000000
ch 3, 388.001000000

>tornado:# wr-dio-cmd wr0 stamp 2
ch 2, 385.001000000
ch 2, 386.001000000
ch 2, 387.001000000
ch 2, 388.001000000

>tornado:# wr-dio-cmd wr0 stamp 4
ch 4, 385.001000000
ch 4, 386.001000000
ch 4, 387.001000000
```
6 Quick Start Guide For Developers

6.1 Introduction

**Notes:** This section is still in development, you can read it to have a few hints on where to find the project but nothing more.

This section has the purpose to give the developers a quick start guide to install the tools and compile the different projects to obtain the binaries used previously.

The starting kit is base on various project:

- **spec-sw** The project that contains the software (application + driver) which you have already compiled above.
- **spec-golden** A simple gateware in order to access to the EEPROM of the FMC (WB-I2C)
- **wr-nic** gateware (FPGA HDL) that includes the NIC & DIO capabilities.
- **wrpc-sw** LM32 sofware in the white rabbit PTP core for the synchronization.

6.2 Tools

You must have installed the following tools:

- Xilinx ISE (>14.x)
- Git
- Build-essentials
- Autoconf/Automake
- Kernel source
- Lm32 cross compiler
- Hdl make
- Texinfo

For all these steps you can follow our guide ohwr_development.pdf
6.3 Golden SPEC gateware

*Dep: hdlmake, Xilinx ISE 14*

This is a really simple gateware that let the PC reads the FMC-EEPROM in order to know what type of gateware it should load. The gateware is a wishbone PCIe bridge connected to a WB I2C module.

To synthetize it just follow the next steps:

```bash
## Checkout the code
svn checkout http://svn.ohwr.org/spec/trunk/hdl/golden@53 spec-golden

## Go to the main directory
cd spec-golden/syn/

## Synthetize using hdlmake
dlmak --fetch
dlmak
make
```

You will therefore obtain your golden firmware called as spec-init.bin

6.4 WRPC-SW (LM32 firmware)

*Notes: The steps in this section are not needed as we already provide the wrc.ram in the wr-nic repository. However we provide here a simple resume of the steps to follow to compile the firmware specifically for the wr-nic. You should also look at the wrpc.pdf to understand how to use it and how to compile for other configurations.*

You can download it from wr-nic-v1.0.tar.gz file or you can try to compile it following the instructions below:

You first need to install the lm32 compiler as suggested in wrpc.pdf, then you need to compile using the specific configuration as bellow:

```bash
## Set up CROSS_COMPILE variable for this terminal
export CROSS_COMPILE="<your_path_to_lm32>/lm32/bin/lm32-elf-";

## Clone the repository
git clone git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git
cd wrpc-sw

## Checkout the stable release
git checkout -b wr-nic-v1.0 wr-nic-v1.0
```

And finally configure & compile it

```bash
## Configuring the project for SPEC
make wrnic_defconfig

## Compile
make
```

You should obtain various files named wrc.bin, wrc.elf, wrc.vhd, wrc.ram

You can therefore use them to override the one in wr-nic project.

```bash
## Override the default embedded wrpc-sw

cp wrc.ram <wr_ROOT_FOLDER>/wr-nic/syn/spec
```
6.5 WR-NIC (HDL-gateware)

This step shows us how to prepare the WR-NIC bitstream (SPEC+FMC DIO) with the wrpc-sw (wrc.ram file) embedded inside.

```bash
## Checkout the code
git clone git://ohwr.org/white-rabbit/wr-nic.git
cd wr-nic

## Checkout the stable release
git checkout -b wr-nic-v1.0 wr-nic-v1.0

## Go to the main directory
cd wr-nic/syn/spec/

## Synthetize using hdlmake
hdlmake --fetch
hdlmake --fetch
hdlmake -l
```

You should finally obtain the bitstream to import in your fmc driver folder.

7 Appendix

Some other quick informations are regrouped in the appendix

7.1 WRPC as grandmaster

In basic configuration your Master SPEC can use its internal free-running oscillator as a time reference. However, you can also discipline your Master SPEC with external 10 MHz and 1-PPS signal by connecting them to the appropriate LEMO connectors of DIO FMC board:

![Figure 13: Grandmaster setup](image)

Then in the wrc console just execute the following commands:
8 FAQS & Bugs

8.1 Known Bugs

8.1.1 Virtual UART

The following command: `sudo tools/spec-vuart -b <dev_id>` does not show anything?

Virtual UART is not enabled on the current release but will be available on the next one. Until the next release you should stick with the Physical UART: micro USB cable and minicom.

8.1.2 No SFP in EEPROM database

When you try to match your SFPs and you obtain an error such as **Could not match to DB** you need to add your own SFP parameters.

```
## List your actual sfp database
wrc# sfp show

## Erase the actual database if it was corrupted
wrc# sfp erase

## Add the parameters for the SFP provided in the starting kit
wrc# sfp add AXGE-1254-0531 46407 167843 73622176
wrc# sfp add AXGE-3454-0531 46407 167843 -73622176
```

You can also check the `wrpc.pdf` for extended explanations on how to obtain the parameters for your own SFP and how to calibrate them.

8.1.3 Modprobe is not working

If you get and error like the following:

```
>:$ sudo modprobe wr-nic
FATAL: Module wr_nic not found.
```

Be sure that you have run the `sudo depmod -a` command. Otherwise you can still load the kernel modules with insmod using the following order:

```
>:$ sudo insmod kernel/fmc.ko
>:$ sudo insmod kernel/spec.ko
>:$ sudo insmod kernel/wr-nic.ko
```

8.1.4 The system hangs-up

Unfortunately, in some circumstances the system may hang during or slightly after firmware loading. We are trying to solve this problem. You can contribute by sending us a log with the following information:
• What is your system & distribution (uname -a)
• All the command executed from login to the crash
• The serial log starting from the login (dmesg)

8.1.5 Only one wrX interface is brought up

On Ubuntu 12.04 (32bits & 64bits), the driver only load one card. This might due to some problems with network manager. We suggest you to reboot your system and restart the operation to solve this issue.

8.2 Frequently Added Questions

8.2.1 I don’t see the interface using ifconfig

Please check if you see them using ifconfig -a, if you see them by using the -a option you just need to bring them up. This is explained in the Bring up the network interface section.

8.2.2 I am stuck in calibration!

You have run

```
wr# calibration force
Enabling ptracker channel: 0
LNK:
PLL: SPLL_Init: running as Slave, 1 ref channels, 2 out channels
```

And the calibration is doing an infinite loop.

Calibration is only available for slave and you need to be connected to a working PTP master (SPEC in master mode or the switch) to be completed. To configure your SPEC as a working PTP master, just execute in the wrpc shell of the other board:

```
wr# sfp match
wr# mode master
wr# ptp start
```

If you have blocked your two boards in calibration mode and they stay in this mode even after rebooting and reloading firmware/gateware you might need to format the EEPROM.

8.2.3 How can I format the EEPROM?

Generate an empty file of 64kb (the extension must be .bin)

```
dd count=64 bs=1k if=/dev/zero of=/tmp/zero64k.bin
```

and use the fmc-write-eprom module from the spec-sw project to flash the eeprom with zero bytes.

```
insmod kernel/fmc-write-eprom.ko file=/tmp/zero64k.bin busid=<busid_SPEC-DIO>
```

**Notes:** By doing this you will lose all the SFP informations and therefore you will need to recalibrate “correctly” your boards.
8.2.4 The link is down, but everything is connected

The SFPs and the optical fiber are plugged but the error message appears Link down -- and you cannot see any green LED on the SPEC.

Please check that the SFP connector is totally plugged into the SPEC. The SFP cage of the SPEC can be blocked sometime by the PC back-panel. You must push the SFP until hearing a “clicking noise”.

8.2.5 What is the latency in the WR-NIC

Many people ask us about the latency in the WR-NIC.

The answer depends really on what you want to measure: The only “relevant” delay might be the time between the SFP and the PCIe which is less than 20 cycles. However the delay introduced by your nondeterministic OS is much more important.

9 References

- spec-sw.pdf main documentation of the spec-sw
- wrpc.pdf White Rabbit Core documentation
- fmc-bus.pdf How the FMC bus driver works and how to write one for your own FMC board.
- SFPs information http://www.ohwr.org/projects/white-rabbit/wiki/SFP
- White Rabbit Calibration: http://www.ohwr.org/documents/213
- spec-2-spec: SPEC-2-SPEC demo