## Contents

1 What is new in wr-nic-v2.0?  2
2 Introduction  2
3 Gateware elements  4
  3.1 WRPC (White Rabbit PTP Core)  6
  3.2 DIO core  6
4 DIO Configuration & Control  9
  4.1 DIO core utilization  9
  4.2 Interrupt handling  11
5 How to synthetize it?  11
  5.1 HDLMake  11
  5.2 WR-NIC (HDL-gateware)  12
  5.3 WRPC-SW (LM32 firmware)  12
6 Software support and applications  13
7 New features: Remote configuration based on Etherbone module  13
8 WR NIC calibration  15
9 Troubleshooting  16
10 Future work  17
11 References  17
1 What is new in wr-nic-v2.0?

In this section, we detail what is new in \texttt{wr-nic-v2.0} comparing with older versions:

1. The Etherbone core has been added in the WR NIC project design to allow remote configuration over the network using TCP/UDP packets.
2. The first DIO channel output is reserved for the 1-PPS signal of WRPC. It has been done because we detected an offset between 1-PPS input signal and 1-PPS output signal for the Grandmaster configuration.
3. The calibration process was performed to fulfill the clock delay restriction of White Rabbit technology (less than 1 ns).
4. The DIO register structure has been changed. A number of bits has been increased in order to define new modes for DIO channels.
5. The WRPC support has been changed to incorporate the following features (see \texttt{wrpc-v2.1-changes}):
   - WRPC-SW:
     - The WRPC packet filter has been configured with new filter rules in order to allow the Etherbone packets reach the Etherbone core and to avoid discarding the incoming packets. A new Kconfig parameter has been defined (\texttt{NIC_PFILTER}) not to interfere with the other packet filter configurations.
     - A new WRPC command has been added (\texttt{refresh}). This command allows to set the update period of gui and stat commands.
     - The Grand-Master lock mechanism has been implemented.
     - The t24p calibration is executed every time the WRPC is set to Slave mode.
     - A new commands have been added to set time manually.
     - The MAC address is set by default if it can not get a value from 1-wire device.
     - PPSi is default configuration instead of ptp-noposix now. It adds compatibility with more PTP devices.
   - WRPC:
     - A new generic parameter has been added to the WRPC interface. the \texttt{g_vuart_fifo_size} allows to set the UART fifo size of WRPC.
     - A separate 1-PPS output has been added for driving LED.
     - The reset module has been improved.
     - A new interface has been added for new t24p procedure.
     - The SoftPLL mode has been updated (multichannel and selectable phase detector mode).

The complete list of new WRPC features can be found in \texttt{wrpc-v2.1-changes}.

2 Introduction

The White-Rabbit Network Interface Card (WR-NIC) project is concerned with the development of gateware to make the combination of a SPEC card and a DIO mezzanine behave as a Network Interface Card (NIC) under Linux. A basic demo is available at the Starting-kit project example that uses two SPEC boards, one configured as Grandmaster and one as slave. Different simple use
cases are provided as basic examples and are available at starting-kit. This document focuses on the description of the project gateware. This manual is part of the associated hardware project, hosted at wr-nic-repository, which git repository hosts the latest version. You can find more information about SPEC card at spec/spec-sw and about FMC DIO card in fmc-dio-5ch.

**NIC project elements**

![Diagram of NIC project elements]

Figure 1: Basic wr-nic project elements

Note that the WR-NIC project inherits many codes and working methodology of many other projects. Specially important to highly the following ones:

1. The White-Rabbit core collection (commit d85255b): wr-cores. Look at the WR PTP Core. It is a black-box standalone WR-PTP protocol core, incorporating a CPU (LatticeMicro 32, LM32), WR MAC and PLLs. It is also convenient to look at TxTSU and NIC project. For further details, search for its related wbgen2 files (extension .wb).
2. Software for White-Rabbit PTP core (commit 9f3bcfa): wrpc-sw (a subproject of the previous one)
3. The Gennum GN4124 core (commit 3844fe5): GN4124. This project shows how to configure and control the gn4124 chip in order to communicate the SPEC carrier through PCIe interface.
4. The platform independent core collection (commit 76f2e5d): general-cores. This project contains several useful IP cores such as memories, synchronizer circuits, uart modules, Wishbone crossbar, etc.
5. The Etherbone core (commit 08b55da): etherbone. This gateware module allows to access Wishbone peripheral memory map using the Ethernet interface. This make possible to read device data as well as read/write configuration registers across the network.
In addition to these projects, software support is provided from the project:

1. Software (driver, fmc-bus and SPEC working examples) spec-sw. This project contains Linux kernel drivers and tool to work with SPEC card. It requires a “golden FPGA gateware” (spec-init.bin) which is available at spec-sw-files.

2. Starting kit tutorial. This project complements the gateware elements here described with the related PC software examples. A quick overview about this project is presented at section 6. The complete project is available at starting-kit.

3 Gateware elements

The main block modules of the architecture are described on next figure.

![Basic on-chip architecture](image)

Figure 2: Main HDL blocks of the wr-nic gateware
Here is a quick description of each block:

- The **DIO core** (Wishbone slave peripheral) allows the configuration of each one of the five channels of the DIO mezzanine as input or output. For inputs, it provides an accurate time stamp (using time information from the WRPC, not shown in the diagram) and optionally a host (PCIe) interrupt via the IRQ Gen block. For outputs, it allows the user to schedule the generation of a pulse at a given future time, or to generate it immediately.

- The **IRQ Gen** block (Wishbone slave peripheral) receives one-tick-long pulses from other blocks and generates interrupt requests to the GN4124 core. It also includes interrupt source and mask registers.

- The **WB intercon** block ensures seamless interconnection of Wishbone masters and slaves using a crossbar topology.

- The **GN4124 core** is a bridge between the GN4124 PCIe interface chip and the internal Wishbone bus, allowing communication with the host and interrupts (pipelined). It is a Wishbone master and therefore able to write system registers.

- The **WRPC (White Rabbit PTP Core)** communicates with the outside world through the SFP socket in the SPEC, typically using fiber optics. It deals with the WR PTP using an internal LM32 CPU running a portable PTP stack. It forwards/receives non-PTP frames to/from the NIC block, using two pipelined Wishbone interfaces (master and slave for forwarding and receiving respectively). It also provides time information to other cores (not represented in the diagram), and time-tags for transmitted and received frames that can be read through Wishbone for diagnostics purposes. The WR-NIC project uses the latest version including the PPSi library for PTPv2 protocol support.

- The **NIC core** ensures communication between the host and the WRPC. More precisely, it interrupts the host and provides a descriptor that the host can use to fetch incoming frames. For outgoing frames, it receives a descriptor from the host, fetches the frame using PCIe via the GN4124 core and sends it to the WRPC using a pipelined Wishbone interface.

- The **TxTSU module** collects timestamps with associated Ethernet frame identifiers and puts them in a shared FIFO (port identifier is also included although not required for the SPEC card because only one Ethernet port is available but it is included to provide a common descriptor with the switch data). A IRQ is triggered when FIFO is not empty so drivers could read TX timestamps and frame/port identifiers.

- The **Etherbone core** is an IP core developed by GSI that allows remote configuration of device memory map with UDP/TCP network packets.

- The **MUX module** is a multiplexer used to route packets from Endpoint to its destination core. Latter is different depending on the packet type. Each packet is classified by one configurable packet processor located in WRPC. This core has a filter rule table and allows to define up to 8 different packet classes. For WR-NIC project, three classes have been defined: Class 7 is for NIC core, Class 5 for Etherbone one and Class 0 for LM32.

In the next sections we provide a little more information about **DIO core** and the **WRPC (White Rabbit PTP Core)** in order to understand better how the complete system works. Finally, it is important to know that current HDL code contains commented code to activate on-chip logic analyzer circuitry for debugging based on Chipscope of Xilinx. Top file as well as different peripherals include the signals TRIG0 - TRIG3 to help on this purpose. Nevertheless, by default they are commented to avoid wasting unnecessary resources (in fact it could be required to reduce blockram utilization, for instance of NIC or wr_core module in order to use Chipscope on the project, otherwise design could be overmapped).
3.1 WRPC (White Rabbit PTP Core)

The WRPC (White Rabbit PTP Core) block is the HDL block that makes possible the White-Rabbit timing functionalities on the White-Rabbit nodes. It is a black-box standalone WR-PTP protocol core, incorporating a CPU LM32, WR MAC and PLLs. It could be configure to work on 3 different operations modes:

- **Grandmaster**: The WRPC locks to an external 10 MHz and PPS signal provided for instance from a Cesium clock / GPS. This signals are provided from the channel 4 (PPS) and channel 5 (10 MHz) lemo connectors of the DIO card.

- **Master**: The systems uses the VCO oscillators of the SPEC board basically on a free running modes.

- **Slave**: The clock information is recovered from the Ethernet connections and the local clock uses the DDMTD method to follow the external clock.

In this project, the WRPC provides the timing information used for accurate output generation and input time stamping of the DIO signals. Note that this data is provided with an accuracy of 16 ns.

Please note that the current gateware contains the LM32 binary firmware (as part of the FPGA). Therefore no additional LM32 upload programming is required if default code is appropriate for your project. The embedded binary corresponds to the `wrpc-v2.1-for-wrnic` release available at: [wrpc-sw-repository](#).

It is important to remark that for this release the I2C bus of the FMC-DIO card is connected to WRPC. This is needed because current implementation of WRPC store configuration data on the FMC-DIO card EEPROM. Please be aware that for future releases this could change.

The whole description of the core goes beyond the scope of this documentation but the additional information is available at: [wrpc](#) and in: [GD-mgr](#).

3.2 DIO core

The DIO Core block is the HDL block that access to the fmc-dio-5ch mezzanine card. The core goes beyond a standard GPIO core to include advanced functionalities related with timing. Their main elements are shown on the next figure.
Notes: In the previous release we define the DIO channel P mode as a DIO channel mode that allows to generate a 1-PPS WRPC signal output. This was possible for all DIO channels. In the current design, channel 0 output is reserved to work on this mode and no other channels can generate this output. For this reason, the P mode has been removed and it is no longer used.

The different submodule descriptions are:

- **GPIO**: GPIO (General Purpose Input/Output) core allows to use a number of pins as digital input/output signal. A GPIO pin can be configured as input/output with/without tri-state buffer and write/read to/from it.

- **I2C**: It allows to set the threshold of the ADCMP604 fast LVDS comparator and to access to write/read data to the EEPROM memory (24AA64). Please note that already commented, current release do not allow to access I2C signals of the FMC-DIO card from this HDL block. This is assigned to WRPC and therefore it should be handled inside its memory maps (concretely, inside WRPC Syscon device). This could change for future releases.

- **OneWire**: It is used for temperature acquisition.

- Modules to generate/stamp pulses:
  
  1. **Pulse generator**: It produces a programmable ticks-long pulse in its output when the time passed to it through a vector equals a pre-programmed time.
2. **Pulse stamper:** It associates a time-tag with an asynchronous input pulse.

- Additional Wishbone slave core generated elements are:
  1. Trigger registers (time counters for pulse generation).
  2. FIFOs to store the timestamps of input signals.
  3. Interrupt control registers that allow to configure the interrupts generated when there are data in the FIFOs.
  4. Monostable register, which generates a single clock cycle-long.
  5. Update the mode of the drivers: the termination resistors and if it is connected to GPIO, DIO core or WRPC.

As we told in earlier section, we changed connections for the channel 0 of the DIO core. For its output, we connected the 1-PPS output signal from the WRPC. In addition, you can configure this channel in order to use it as user input channel but its output is reserved to PPS signal.

![Figure 4: PPS signal output for channel 0 of DIO](image)
4 DIO Configuration & Control

Accessing to the different system elements is as simple as doing a read/write memory access. The memory map of the different elements of the board are shown in the next figure.

![Memory map diagram](image)

**Figure 5: Memory map**

Any address within this memory space may be addressed by the PC to configure corresponding module (WRPTP, NIC, T xTSU, etc...). Many of them are already well described in related project so we skip their descriptions to focus on new additions. In next paragraphs we just focus on the DIO core which is the new block added to the OHWR and required for this project.

4.1 DIO core utilization

The DIO core, according to its architecture already shown, it allows to read input data of each of the 5 channel with precise time-tag information provided by the WRPTP core. It is also possible to program output at a precise time or we could just generate an output signals immediately. In addition, it is also possible to configure different boards elements as terminator resistors or reference voltage Level using the DAC. All these elements could be controlled independently for each of the 5 channels. More information about the different board configuration elements is available at: [fmc-dio-5ch](#). Note that dio channels time base work with 16 ns accuracy for inputs time-tagging. Outputs need to be generated align on 16 ns time-slices but their time-stamps values have subnanosecond accuracy thanks to the White-Rabbit timing properties.

In order to use input/output channels as previously described, the following actions are required:

- The I/O mode of each channel controlled by the `dio_iomode` register:
  - [0-1]: The two first bit correspond to which signal its connected: 0 (00) GPIO, 1 (01) DIO core, 2 (10) Clock, 3 (11) Undefined (For future use, this value is not used yet).
– [2]: Output Enable Negative (Input enable).
– [3]: 50 Ohm termination enable.

1. 0x0 (PPS Output without termination)
2. 0x0 (GPIO out without termination)
3. 0x0 (…)
4. 0x4 (Input without termination)
5. 0x6: Clock\(^1\) input without termination.

• **Time-programmable pulse generation**: Generate a programmed output at any time at channel X (X between 1 and 4 identifies the requested channel, the channel 0 is reserved for PPS output). For this purpose you need to perform the following actions:

  – Set the required time. This means to provide the 40 bits for the time value and the number of cycles (28 bits). This requires to write the registers `dio_trigX_seconds`, `dio_trighX_seconds` (high part of the time value) and `dio_cyc0_cyc`.
  – Checking if the board is ready for accepting new triggers. This can be done by reading a 1 found at each bit of `dio_trig_rdy` register. The EIC bits 9 to 5 have associated interrupts (active means system is ready to accept new trigger values but please check you have properly configured EIC interrupt mask before). Both methods are possible to check the status. Nevertheless note that the non-ready periods are very shorts (13 cycles of a 62.5 MHz clock, 208 ns) so systems is almost always ready for new trigger values.
  – Arming the trigger. You need to write a 1 at the corresponding bit of the `dio_latch_time_chX` bit field.
  
  After these operations, an output with the programmed tick-length will be presented on the desired channel at the requested time. It is not necessary to do software reset to the register.

• **Immediate pulse generation**: An immediate pulse is generated at the output of each of the card channels just by writing a corresponding 1 at the bit field `dio_pulse_imm_X` when output mode is set to programmable outputs. No reset is required.

• **Variable pulse length**: both output modes, time-programmable as well as immediate allows to configure the output length. By writing the value of the registers `dio_progX_pulse_length` the width of the output pulse could be controlled. The register uses the 28 low significant bits and allow a time duration equal to `register_value x 16 ns`.

• **Input time-tagging**: for each of the 5 inputs, if a 1 is detected at this channel, a precise time information is stored on logic FIFOs including the 40 bits time counters and 28 bits more for the cycles (fifo depth is 256 each one). Currently this information is collected even for pins configured in output mode, GPIO, immediate or time-programmable configurations but it is straightforward to change this at the HDL code. For accessing this information you need to read `dio_tsfX_tag_seconds` (32 low bits), `dio_tsfX_tag_secondsh` (high bits), `dio_tsfX_tag_cycles`. Each time the time tag of any channel is stored, the `fifo not empty` flag generates an interruption to the PC. In the next section we describe these mechanisms.

A detailed information about the memory maps and related registers names are available by generating html documentation of the different wishbone slaves. Download the related .wb file and generate the HTML documentation using wbgen2 tool (for instance `wbgen2 -D diocore.htm wr_nic.wb`).

\(^1\)The Clock input is similar the DIO input excepting that no interrupt will be generated to the PCIe core. It is used in Grandmaster mode as 10 MHz input channel.
4.2 Interrupt handling

The VIC module block is in charge of handling the different interrupts and provide proper registers to inform of the source of each interruption. The main interrupt signal is communicated to the PC using the gn4124 chip and gn4124 and GPIO-8. A proper core in the FPGA uses the irq_req_p1_i signal to activate this external GPIO pin which needs to be assigned at the low-level hardware.

The base address, as shown on the memory map figure is 0x00060000. It handles the following interrupts sources:

- TxTSU interrupts $\rightarrow$ at source 0.
- WRSW-NIC interrupts $\rightarrow$ at source 1.
- DIO-core interrupts $\rightarrow$ at source 2.

Low sources have the highest priority. In order to check the register layout, get the HTML help from the wb_vic.wb file. Information about VIC control and configuration registers are provided there. Because VIC module is done to cooperate with wbgen2 peripherals Embedded Interrupts controllers (EICs), the related information should also be checked. Basically, VIC inform about the main interrupt source and then we need to check the wishbone peripheral interrupt register to complete the interrupt information. Please note that each peripheral generating interrupts has own interrupts registers so a proper configuration of them is required to set-up the interrupt operation.

For instance for the DIO core, please check the status of interrupt registers by looking at wr_nic.wb as previously described.

5 How to synthetize it?

Fetching source files from the repositories, synthetize and download into the FPGA are the target actions. Additionally, in the WR-NIC project you will find a wrc.ram file which corresponds to the LM32 firmware that we are going to embedded into our HDL gateware. It is not needed to recompile it, however if you want to modify the lm32 behaviour you can look at the wrpc-sw-lm32-firmware section to check how to build it.

5.1 HDLMake

In order to synthetize the HDL for wr-nic you need to have installed the hdlmake tool. For more information on how to install and use it, you can look at the hdlmake

To generate the wr-nic bitstream we have used latest commit in the isyp branch.

```
cd <path-hdl-make>
git checkout 6b7aa78
```

Notes: the synthetization can work with other hdlmake verison but if you obtain any errors we recommend you to use the same version as ours.
5.2 WR-NIC (HDL-gateware)

This step shows 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
git checkout -b wr-nic-v2.0 wr-nic-v2.0

## Create and update the submodules
git submodule init
git submodule update

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

## Synthetize using hdlmake
hdlmake --make-ise --ise-proj
make
```

**Notes:** To use the generated gateware with the spec-sw package you should rename it (wr_nic_dio_top.bin) to wr_nic_dio.bin and place it in the /lib/firmware/fmc folder.

5.3 WRPC-SW (LM32 firmware)

This step is not necessary because a precompiled wrc.ram is already provided (in file section of wr-nic). However, if you want to recompile it, you should follow instructions below:

You should first installed the lm32 compiler as suggested in wrpc-v2.0.pdf documentation located at attachment section in wrpc, then you can compile executing the following commands:

```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
$ git checkout -b wrpc-v2.1-for-wrnic wrpc-v2.1-for-wrnic

And finally configure & compile it

# Configuring the project for SPEC with Etherbone support
$ make wrnic_defconfig

# Compile
$ make
```

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

You can therefore try to override it and go back to section wr-nic-hdl-gateware.
# Override the default embedded wrpc-sw

cp wrc.ram <wr_root_folder>/wr-nic/syn/spec

Notes: These steps are a simple summary on how to compile the firmware specifically for the wr-nic, you should also look at the wrpc-v2.0.pdf documentation located at attachments section in wrpc to understand how to use it and how to compile for other configurations.

6 Software support and applications

This project could be used as starting demo with White-Rabbit technology, illustrating its timing capabilities. As already described, the current configuration allows to transform SPEC board on a Network Interface Card with White-Rabbit capabilities. In order to use it like this, some additional software is required:

- SPEC driver supporting the DIO card functionalities already described.
- Applications examples. Starting kit project documentation has several examples that can be probed with DIO FMC card. Some examples are:
  - Configuring of each DIO FMC channel as Input or Output.
  - Generating of immediate/programmable pulses in a DIO FMC Channel.
  - Grandmaster example and 1-PPS generation.

Both elements are described in the software manual of the WR-NIC project (Starting kit) and it is out of the scope of current document to describe them with further details starting-kit. Please read that document in order to have a complete understanding of the NIC project.

7 New features: Remote configuration based on Etherbone module

The Etherbone core is an IP core developed by GSI that allows to read/write in device memory map remotely. To achieve this goal, the Etherbone project provides a software network library to send TCP/UDP configuration packets. The Etherbone core receives these packets and performs needed configuration actions. If you want to get more information of the Etherbone project, visit etherbone.

WR NIC project has been designed to work in a PCI co-processor configuration and as a standalone device. If you use it as a co-processor, there is no problem in configuring it because you can use the kernel drivers developed (see in spec-sw). However, if you want to use it in standalone mode, it can not be accessed directly by PCI bridge since the spec-sw kernel drivers do not solve this problem. You need an access based on network packets. The Etherbone project provides this mechanism. That is why the Etherbone core has been added in the new release of the WR NIC project.

Two things are needed in order to the incoming packets reach the Etherbone core:

- WRPC has an filter packet processor (IP core) to decide which packets are accepted and which are discarded. Packets are classified in one class chosen between eight different classes.
This class stores information about to where a packet must be delivered. So, we have to modify filter rules (see dev/ep_pfilter.c in wrpc-sw-repository) in order to accept Etherbone traffic. We have assigned class 5 to distinguish it from another type of traffic. Moreover, DROP instruction has been removed in order to avoid any packet to be discarded (because a conventional NIC receives all type of packages). We have used a NIC_PFIL FILTER macro not to interfere with configurations needed for other projects.

```c
#define R_CLASS(x) (24 + x)
#define R_DROP 23

void pfilter_init_default()
{
    pfilter_new();
    ...

#ifdef CONFIG_ETHERBONE
    /* r10 = IP(unicast) */
    pfilter_logic3(10, 3, OR, 0, AND, 4);

    /* r11 = IP(unicast+broadcast) */
    pfilter_logic3(11, 1, OR, 3, AND, 4);

    /* r14 = ARP(broadcast) or PTPv2 */
    pfilter_logic3(14, 1, AND, 6, OR, 5);

    /* r15 = ICMP/IP(unicast) or ARP(broadcast) or PTPv2 */
    pfilter_logic3(15, 10, AND, 7, OR, 14);

    /* Ethernet = 14 bytes, IPv4 = 20 bytes, offset to dport: 2 = 36/2 = 18 */
    /* r14 = 1 when dport = BOOTPC */
    pfilter_cmp(18, 0x0044, 0xffff, MOV, 14);

    /* r14 = BOOTP/UDP/IP(unicast|broadcast) */
    pfilter_logic3(14, 14, AND, 8, AND, 11);

    /* r15 = BOOTP/UDP/IP(unicast|broadcast) or ICMP/IP(unicast) or ARP(broadcast)
        or PTPv2 */
    pfilter_logic2(15, 14, OR, 15);

#ifdef CONFIG_NIC_PFILTER
    /* r6 = 1 when dport = ETHERBONE */
    pfilter_cmp(18, 0xebd0, 0xffff, MOV, 6);

    /* r9 = 1 when magic number = ETHERBONE */
    //pfilter_cmp(21,0x4e6f,0xffff,MOV,9);
    //pfilter_logic2(6,6,AND,9);

    /* class 0: ICMP/IP(unicast) or ARP(broadcast) or PTPv2 => PTP LM32 core */
    pfilter_logic2(R_CLASS(0), 15, MOV, 0);

    /* class 5: Etherbone packet => Etherbone Core */
#endif
#endif
```
pfilter_logic2(R_CLASS(5), 6, OR, 0);

/* class 7: Rest => NIC Core */
pfilter_logic3(R_CLASS(7), 15, OR, 6, NOT, 0);

#else
...

pfilter_load();

• A route mechanism to reach Etherbone core must be added. It is implemented with a MUX IP core component that can route one packet depending on its class. We have placed one MUX between WRPC (fabric interface) and NIC. So, packets from WRPC fabric interface can reach the NIC or the Etherbone core.

Thanks to the Etherbone core and its software library, we can access the device using network packets. One interesting application is to allow access in the same way we connect to it via the USB serial port. University of Granada has developed a high-level library based on Etherbone that allows you to define devices configuration files with this associated operations. This files can be loaded with CALoE methods and user can evoke device operations as C functions (using CALoE API of course). This library is a alpha version and you can find it in starting-kit repository (ugr branch).

8 WR NIC calibration

The WR NIC calibration process is necessary to fulfill clock delay restriction of White Rabbit technology. It tries to find temporal parameters in order to compensate transmission/reception delays of gateware and asymmetry of fiber link.

The calibration process has two steps. The first of them consists in pre-calibrating a device to use it as reference for all other devices in your White Rabbit network. The pre-calibration process requires two identical WR devices (board, gateware and software) to make the assumption that the time delays are equal and thus simplifying the procedure.

Second step is the calibration of all other devices in your network. To do it, you have to use the pre-calibrator of previous step. The WR calibration process is described in detail in wr-calibration.

We have needed the following equipment in order to calibrate:

• An oscilloscope (RIGOL DS 6062).
• Two SPEC v4 boards with DIO FMC card (same gateware and software is required for pre-calibration process).
• A WR Switch (18 ports, HW: 3.3, GW: 3.3). You can find the WR Switch v3.3 calibration parameters used by us in wr-calibration.
• Two LEMO links (1 meter, they should be same length or you must be know their time delays to compensate the error they introduce).
• A LEMO-BNC adapter.
There are some considerations about the gateware properties that need to be well understood in order to avoid problems. They are:

- Properly setting of interrupts registers or wrong memory maps are the typical errors at this stage. Please check it carefully.

- A WRPC firmware is provided inside the WR-NIC bitstream file (wrc.ram file on the project folder). However, WRPC firmware can be replaced by another version compiled by the user. The replacement can be made during synthesis process (update wrc.ram file before) or using spec-cl loader (see spec-sw project).
• The programmable output does not support buffering mode. Therefore, if one output is pro-
grammed, user should avoid to reprogramming until output pulse has been done. Otherwise,
previous pulse will be lost. This is implemented as it to simplify the hardware and speci-
fications (current functionality works well for our simple illustrative applications examples).
Nevertheless it should be taken into account if you develop your own application.

• Timestamping granularity of inputs DIO channel is limited to 16 ns so there is not any error
if further accuracy is not obtained. Nevertheless, note White-Rabbit will still synchronize the
system clock with subnanosecond accuracy.

Further information will be provided in future releases.

10 Future work

• Nowadays, the WR NIC maximum bandwidth is about 11 Mbps because DMA module is not
included in the design. As future work, we will develop a DMA module in order to increase
the wr-nic performance.

11 References

• G. Daniluk, White Rabbit PTP Core the sub-nanosecond time synchronization over Ethernet,
2012. M.Sc thesis describing the development and implementation of the first standalone HDL
module handling the sub-nanosecond synchronization over a regular Ethernet - the White
Rabbit PTP Core. Available at: GD-mgr
• Open Hardware Repository website at ohwr
• White Rabbit NIC project at wr-nic
• Starting kit project at starting-kit
• White Rabbit cores project at wr-cores
• White Rabbit PTP core project at wrpc
• Software for White Rabbit PTP core at wrpc-sw
• Gennum GN4124 core project at GN4124
• Platform independent cores at general-cores
• Etherbone project at etherbone
• SPEC card project at spec
• Software support for SPEC card at spec-sw
• FMC DIO card of 5 channels at fmc-dio-5ch
• White Rabbit procedure calibration at wr-calibration
• HDLmake tool project at hdlmake
• WRPC release 2.1 changes at wrpc-v2.1-changes