White Rabbit PTP Core User's Manual

August 2012 – Release 2.0
Building and Running

Grzegorz Daniluk (CERN BE-CO-HT)

Table of Contents

Introduction

This is the user manual for the White Rabbit PTP Core developed on `ohwr.org`. It describes the building and running process. If you don't want to get your hands dirty and prefer to use the binaries available at `http://www.ohwr.org/projects/wr-cores/files` you can skip Chapter 2 [Building the Core], page 1 and move forward directly to Chapter 3 [Running and Configuring], page 4.

1 Repositories and Releases

This version of the White Rabbit PTP Core is release 2.0. The code and documentation is distributed in the following places:

`http://www.ohwr.org/projects/wr-cores/documents`
: This place hosts the pdf documentation for every official release.

`http://www.ohwr.org/projects/wr-cores/files`
: Here we place the *.tar.gz* file for every release, including the *git* tree and synthesized/compiled binaries

`git://ohwr.org/hdl-core-lib/wr-cores.git`
: Read-only repository with the complete HDL design of WRPC

`git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git`
: Read-only repository with the WRPC LM32 software (incl. WR PTP daemon)

Other tools useful in building and running WRPC can be fetched from the following locations:

`git://ohwr.org/misc/hdl-make.git`
: *hdlmake* is used in the HDL synthesis process to build the Makefile based on the set of Manifest files.

`http://www.ohwr.org/attachments/download/1133/lm32.tar.xz`
: LM32 toolchain used to compile the WRPC firmware (software).

The repositories containing the WRPC gateware and software (*wr-cores*, *wrpc-sw*) are tagged with `wrpc-v2.0` tag. Other tools used to build the core and load it into SPEC board should be used in their newest available versions stored in master branch of an appropriate git repository (unless specified otherwise in this document).

Any official hot fixes, if any, for this release will live in the branch called `wrpc-v2.0-fixes`, in each WRPC repository.

2 Building the Core

Building the White Rabbit PTP Core is a two step process. First you have to synthesize the FPGA firmware (gateware). This describes the hardware inside FPGA that is later used by LM32 software to perform WR synchronization.

To perform the steps below you will need a computer running Linux.

2.1 HDL synthesis

Before running the synthesis process you have to make sure that your environment is set correctly. You need the Xilinx ISE Software with free WebPack license installed on a PC. It contains the script *settings32.sh* and *settings32.csh* (depending on the shell you use) that sets up all the system variables required by Xilinx software. For default installation path the script is located in:

```
/opt/Xilinx/<version>/ISE_DS/settings32.sh
```

and has to be executed before other tools are used. The easiest way to ensure that *ISE*-related variables are set in the shell is to check if *$XILINX* variable contains the path to your *ISE* installation directory.

Note: current version of *hdlmake* tool developed at CERN requires modification of *$XILINX* variable after *settings32* script execution. This (provided that the installation path for *ISE* is /opt/Xilinx/<version>) should look like this:

```
$ export XILINX=/opt/Xilinx/<version>/ISE_DS
```

Note: the Xilinx project file included in the WRPC sources was created with Xilinx ISE 14.1. It is recommended to use the newest available version of ISE software.

HDL sources for WR PTP Core can be synthesized with nothing more but Xilinx ISE software, but using *hdlmake* tool developed at CERN is much more convenient. It creates a synthesis Makefile and ISE project file based on the set of Manifest.py files deployed among directories inside *wr-cores* repository.

First, please clone the *hdlmake* repository from its location given in Chapter 1 [Repositories and Releases], page 1:

```
$ git clone git://ohwr.org/misc/hdl-make.git <your_hdlmake_location>
```

and add the *hdlmake* binary location to your *$PATH* to be able to call it from any directory:

```
$ export PATH=<your_hdlmake_location>:$PATH
```

Note: the *hdlmake* usage instructions here are based on version 493ce82. When there will be newer commits, they can be used but please be aware that its execution parameters may change. In that case please refer to *hdlmake* documentation.

Having Xilinx ISE software and *hdlmake* you can clone the main WR PTP Core git repository and start building the FPGA bitstream. First, please create a local copy of the *wr-cores* in the preferred location on your system. This release is marked with wrpc-v2.0 tag.

```
$ git clone git://ohwr.org/hdl-core-lib/wr-cores.git <your_wrpc_location>
$ cd <your_wrpc_location>
$ git checkout wrpc-v2.0
```

Note: alternatively you can get the release sources from the tarball available in `http://www.ohwr.org/projects/wr-cores/files`

The subdirectory which contains the main synthesis Manifest.py for SPEC board and in which you should perform the whole process is:

```
$ cd <your_wrpc_location>/syn/spec_1_1/wr_core_demo/
```

Executing *hdlmake* without any parameters will fetch other git repositories containing submodules essential for WRPC, and store their local copies to:

```
<your_wrpc_location>/ip_cores
```

After that, the actual synthesis is just the matter of executing the command:

```
$ make
```

just as in a regular software compilation process. This takes (depending on your computer speed) about 15 minutes and if you are lucky (i.e. there were no errors) it should create FPGA firmware in two files: *spec_top.bit* and *spec_top.bin*. The former can be downloaded to FPGA using Xilinx software (*Impact* or *Chipscope Pro*) and Xilinx Platform Cable. The latter can be used to program the Spartan 6 chip on SPEC using the kernel driver from *spec-sw* repository (usage example in Chapter 3 [Running and Configuring], page 4).

If, on the other hand, you would like to clean-up the repository and rebuild everything from scratch you can use the following commands:

- *$ make clean* - removes all synthesis reports and log files;
- *$ make mrproper* - removes spec_top.bin and spec_top.bit files;
- *$ hdlmake clean* - removes all fetched repositories (modules) from *ip_cores* subdirectory.

2.2 LM32 software compilation

To compile the LM32 software for White Rabbit PTP Core you will need to download and unpack the LM32 toolchain from the location mentioned already in Chapter 1 [Repositories and Releases], page 1:

```
$ wget http://www.ohwr.org/attachments/download/1133/lm32.tar.xz
$ tar xJf lm32.tar.xz -C <your_lm32_location>
```

Similar as with *hdlmake* in Chapter 2 [Building the Core], page 1, you will need to add the LM32 toolchain binaries location to you *$PATH* to be able to call them from any directory:

```
$ export PATH=<your_lm32_location>/lm32/bin:$PATH
```

To get the release sources of WRPC software please clone the *wrpc-sw* git repository tagged with wrpc-v2.0 tag:

```
$ git clone git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git <your_wrpcsw_location>
$ cd <your_wrpcsw_location>
$ git checkout wrpc-v2.0
```

Note: alternatively you can get the release sources from the tarball available in `http://www.ohwr.org/projects/wr-cores/files`

The WRPC software repository contains a ptp-nosposix (that contains the WR PTP software daemon) in the form of a git submodule. Your fresh local copy cloned from *ohwr.org* has therefore the *ptp-noposix* directory empty. To fetch the ptp-noposix you have to execute the following git commands:

```
$ git submodule init
$ git submodule update
```

First you have to compile the tools provided with WRPC software which are used later during the software compilation:

```
$ cd tools
$ make
$ cd ..
```

Now you have everything that is needed to build the software for WRPC. Before compilation the decision can be made whether to turn on or not the software support for Etherbone core that is integrated inside WRPC gateware for SPEC board. By default it is disabled but can be turned on by setting the value of *WITH_ETHERBONE* variable inside the Makefile. The compilation is made by a simple command without any additional parameters:

```
$ make
```

The resulting binary *wrc.bin* can be then used with the loader from *spec-sw* software package to program the LM32 inside the White Rabbit PTP Core (Chapter 3 [Running and Configuring], page 4).

3 Running and Configuring

3.1 Downloading firmware to SPEC

There is a Software support for the SPEC board project in *ohwr.org*. It contains a set of Linux kernel drivers and userspace tools written by Alessandro Rubini and Tomasz Wlostowski that are used to communicate with the SPEC board plugged into the PCI-Express port of the PC.

The instructions in this section are based on commit 27b4ad9 of *spec-sw* repository and are limited to absolutely minimum required to load WRPC FPGA and LM32 firmware. The full manual for *spec-sw* can be found on: http://www.ohwr.org/attachments/download/1506/spec-sw-2012-08-08.pdf. If there will be a newer version of SPEC software support you would like to use, the up-to-date documentation can always be found in *doc/* subdirectory of *spec-sw* git repository.

First, please clone the git repository of SPEC software support package and build the kernel driver and userspace tools:

```
$ git clone git://ohwr.org/fmc-projects/spec/spec-sw.git <your_specsw_location>
$ cd <your_specsw_location>
$ git checkout 27b4ad9
$ make
```

Then you have to copy the *spec_top.bin* to /lib/firmware/fmc/. changing its name:

```
$ cp <your_wrpc_location>/syn/spec_1_1/wr_core_demo/spec_top.bin \
     /lib/firmware/fmc/spec-demo.bin
```

and after that you are ready to load the *spec.ko* driver that configures the Spartan 6 FPGA on SPEC with a given bitstream (make sure you are in <your_spacsw_location>:

```
$ insmod kernel/spec.ko name=demo
```

To check if the FPGA firmware file was found by the driver and correctly loaded to FPGA the *dmesg* Linux command can be called. Among plenty of messages you should be able to find something very similar to:

```
  [99883.768214] spec_probe (device 0003:0000)
  [99883.768220] spec_probe: current 8639 (insmod)
  [99883.768248] spec 0000:03:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16
  [99883.768302] spec 0000:03:00.0: irq 49 for MSI/MSI-X
  [99883.768971] spec_load_files
  [99883.774842] spec_load_fpga: got binary file "fmc/spec-demo.bin",
1485512 (0x16aac8) bytes
  [99883.966491] spec_load_submodule: load "fmc/spec-demo": 256
```

If everything went right up to this moment you can write the LM32 binary (*wrc.bin*) to the SPEC board. For this purpose, there is a *spec-cl* tool in the *spec-sw* repository. Programming is done with the simple command below:

```
$ tools/spec-cl <your_wrpcsw_location>/wrc.bin
```

Now you should be able to start the Virtual-UART software (also a part of *spec-sw* package) that will be used to interact with the White Rabbit PTP Core Shell:

```
$ tools/spec-vuart
```

If you are able to see the WRPC Shell prompt *wrc#* that means the Core is up and running on your SPEC. Congratulations !

3.2 Writing EEPROM and calibration

By default WRPC starts in WR Slave mode, uses the calibration values for Axcen AXGE-3454-0531 SFP and for release FPGA bitstream available in *http://www.ohwr.org/projects/wr-cores/files*. This might be fine for running White Rabbit PTP Core for the first time and synchronizing it to WR Switch. There are however, two mechanisms that are useful when playing more with WRPC shell and different settings.

Note: the examples below describe only a subset of WRPC Shell commands required to make a basic configuration and calibration. A full description of all supported commands can be found in Appendix A [WRPC Shell commands], page 9.

First, before making the configuration changes, it is recommended (but not obligatory) to stop the PTP daemon. Then, the debug messages from daemon would not show up to the console while you will interact with the shell.

```
wrc# ptp stop
```

If your SPEC has any Mezzanine board plugged into the FMC connector (e.g. DIO, Fine Delay, TDC...) then you can create a calibration database inside the FMC EEPROM. The example below presents the WRPC Shell commands which create an empty SFP database and add two Axcen transceivers with deltaTx, deltaRx and alpha parameters associated with them. Those SFPs are most widely used in WR development and demonstrations:

```
wrc# sfp erase
wrc# sfp add AXGE-1254-0531 46407 167843 73622176
wrc# sfp add AXGE-3454-0531 46407 167843 -73622176
```

To check the content of the SFP database you can execute the *sfp show* shell command.

The calibration procedure of WRPC is limited to absolutely minimum and is fully automatized. It measures the t2/t4 phase transition point and stores the value into the FMC EEPROM so that the calibration would not have to be repeated every time the Core starts. However, it is ***important*** to remember that this calibration function should be executed only once but for ***every*** new FPGA firmware synthesized form *wr-cores* repository:

```
wrc# ptp stop
wrc# sfp detect
wrc# sfp match
wrc# calibration force
```

The example above detects the SFP transceiver plugged into the SPEC board, tries to read its parameters from our newly created SFP database and forces the calibration to be executed. The *force* argument is required since *calibration* without any arguments tries first to read the t2/t4 phase transition stored in EEPROM, and does not run the procedure if the value was previously stored there. That is used to get the measured value and pass it to PTP daemon before it starts (see init script examples).

The WR PTP Core's mode of operation (WR Master/WR Slave) can be set using the *mode* shell command:

```
wrc# mode slave
```

```
or
  wrc# mode master
```

This stops the PTP daemon, changes the mode of operation, but does not start it back automatically. Therefore after changing it you need to start the daemon manually:

```
wrc# ptp start
```

One option is to type all those commands to initialize the WRPC software to the required state every time the Core starts. However, you can also write your own init script to FMC EEPROM and WRPC software will execute it each time it comes back from the reset state (this also includes coming back from reset after programming the FPGA and LM32). Building the simple script that reads the detected SFP parameters and t2/t4 phase transition value from EEPROM, configures the mode of operation to WR Slave and starts the PTP daemon is presented here:

```
wrc# init erase
wrc# init add ptp stop
wrc# init add sfp detect
wrc# init add sfp match
wrc# init add calibration
wrc# init add mode slave
wrc# init add ptp start
```

Almost exactly the same one can be used for running SPEC in WR Master mode. The only difference would be of course *init add mode slave* vs. *init add mode master*.

3.3 Running the Core

Having the SFP database, t2/t4 phase transition point and the init script created in Section 3.2 [Writing EEPROM and calibration], page 5 you can restart the WR PTP Core by reprogramming the LM32 software (with *spec-cl* tool) or by typing the shell command:

```
wrc# init boot
```

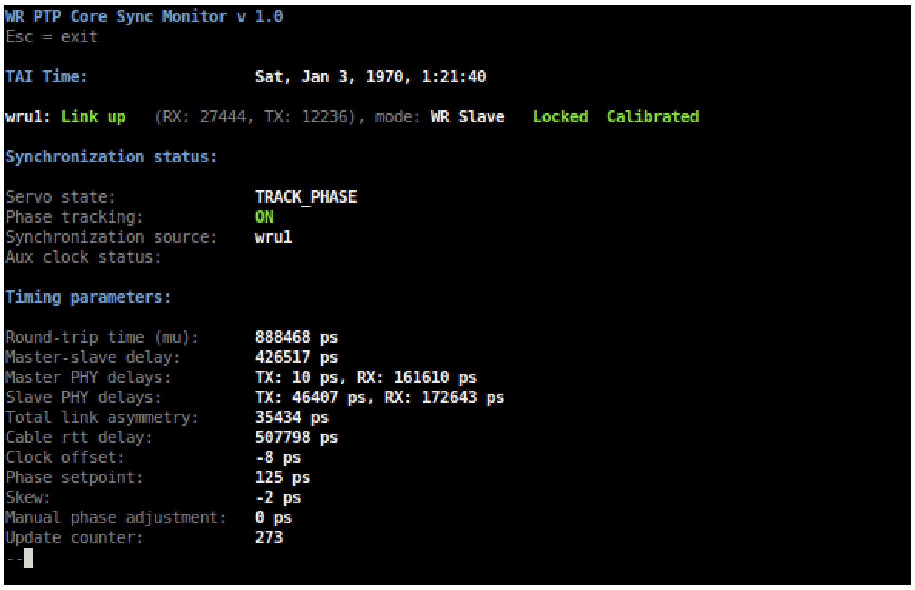
After that you should see the log messages that confirm the init script execution:

```
(...)
executing: ptp stop
executing: sfp detect
AXGE-3454-0531
executing: sfp match
SFP matched, dTx=46407, dRx=167843, alpha=-73622176
executing: calibration
Found phase transition in EEPROM: 2384ps
executing: mode slave
SPLL_Init: running as Slave, 1 ref channels, 2 out channels
Locking PLL
executing: ptp start
wrc# SPLL_Init: running as Slave, 1 ref channels, 2 out channels
Enabling ptracker channel: 0
(...)
```

Now you should have the White Rabbit PTP Core running in WR Slave mode. The Shell also contains the monitoring function which you can use to check the WR synchronization status:

```
wrc# gui
```

The information is presented in a clear, auto-refreshing screen:



```
WR PTP Core Sync Monitor v 1.0
Esc = exit

TAI Time:                  Sat, Jan 3, 1970, 1:21:40

wru1: Link up   (RX: 27444, TX: 12236), 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):      888468 ps
Master-slave delay:        426517 ps
Master PHY delays:         TX: 10 ps, RX: 161610 ps
Slave PHY delays:          TX: 46407 ps, RX: 172643 ps
Total link asymmetry:      35434 ps
Cable rtt delay:           507798 ps
Clock offset:              -8 ps
Phase setpoint:            125 ps
Skew:                      -2 ps
Manual phase adjustment:   0 ps
Update counter:            273
--
```

Note: the *Synchronization status* and *Timing parameters* in *gui* are available only in WR Slave mode. When running as WR Master, you would be able to see only the current date and time, link status, Tx and Rx packet counters, lock and calibration status.

If you have a DIO Mezzanine board placed on your SPEC, you can check the synchronization quality by observing the difference between 1-PPS signals from the WR Master and WR Slave. White Rabbit PTP Core generates 1-PPS signal to the LEMO connector No. 1 on DIO Mezzanine. However, please remember to use oscilloscope cables having the same length and type (with the same delay), or take their delay difference into account in your measurements.

4 Troubleshooting

My computer hangs on loading spec.ko driver.

This will occur when you try to load the *spec.ko* kernel driver while your *spec-vuart* is running and trying to get messages from Virtual-UART's registers inside WRPC. Please remember to quit *spec-vuart* before reloading the driver.

I want to synthesize WRPC but hdlmake does nothing, just quits without any message.

Please check if you have the Xilinx ISE-related system variables set correctly (*settings32.sh* script provided by Xilinx sets them) and make sure you have overwritten the *$XILINX* variable to:

```
$ export XILINX=/opt/Xilinx/<version>/ISE_DS
```

or similar, if your installation folder differs from default.

WR PTP Core seems to work but I observe on my oscilloscope that the offset between 1-PPS signals from WR Master and WR Slave is more than 1 ns.

Run the t2/t4 phase transition value measurement procedure from the WRPC Shell:

```
wrc# ptp stop
wrc# calibration force
```

and check if the oscilloscope cables you use have the same delays (or take the delay difference into account in your measurements).

I can see in the WRPC GUI that the servo cannot reach TRACK_PHASE state.

Please stop the PTP daemon on your SPEC, read your SFP's parameters from SFP database you have created in EEPROM and run the t2/t4 phase transition value measurement procedure form WRPC Shell:

```
wrc# ptp stop
wrc# sfp detect
wrc# sfp match
wrc# calibration force
```

5 Questions, reporting bugs

If you have found a bug, you have problems with White Rabbit PTP Core or one of the tools used to build and run it, you can write to our mailing list `white-rabit-dev@ohwr.org`

Appendix A WRPC Shell Commands

`pll init <mode> <ref_channel> <align_pps>`	manually run spll_init() function to initialize SoftPll
`pll cl <channel>`	check if SoftPLL is locked for the channel
`pll sps <channel> <picoseconds>`	set phase shift for the channel
`pll gps <channel>`	get current and target phase shift for the channel
`pll start <channel>`	start SoftPLL for the channel
`pll stop <channel>`	stop SoftPLL for the channel
`pll sdac <index> <val>`	set the dac
`pll gdac <index>`	get dac's value
`gui`	starts GUI WRPC monitor
`stat`	prints one line log message
`stat cont`	prints log message for each second (Esc to exit back to shell)
`ptp start`	start WR PTP daemon
`ptp stop`	stops WR PTP daemon
`mode`	prints available WR PTP modes
`mode grandmaster`	sets WRPC to operate as Grandmaster clock (requires external 10MHz and 1-PPS reference)(*)
`mode master`	sets WRPC to operate as Free-running Master(*)
`mode slave`	sets WRPC to operate as Slave node(*)
`calibration`	tries to read t2/4 phase transition from EEPROM, if not found runs calibration procedure
`calibration force`	starts calibration procedure that measures t2/4 phase transition, and stores the result to EEPROM
`time`	prints current time from WRPC
`time raw`	prints current time in a raw format (seconds, nanoseconds)
`time set <sec> <nsec>`	sets WRPC time
`sfp detect`	prints the ID of currently used SFP transceiver
`sfp erase`	cleans the SFP database stored in FMC EEPROM
`sfp add <ID> <deltaTx> <deltaRx> <alpha>`	stores calibration parameters for SFP to the database in FMC EEPROM
`sfp show`	prints all SFP transceivers stored in database
`sfp match`	tries to get calibration parameters from database for currently used SFP transceiver(**)

`init erase`	cleans initialization script in FMC EEPROM
`init add <cmd>`	adds shell command at the end of initialization script
`init show`	prints all commands from the script stored in EEPROM
`init boot`	executes the script stored in FMC EEPROM (the same action is done automatically when WRPC starts after resetting LM32)
`mac get`	prints WRPC's MAC address
`mac getp`	re-generates MAC address from 1-wire digital thermometer or EEPROM
`mac set <mac>`	sets the MAC address of WRPC
`mac setp <mac>`	sets MAC address to the 1-wire EEPROM (if available)
`sdb`	prints devices connected to the Wishbone bus inside WRPC
`ip get`	prints the IPv4 address of the WRPC(***)
`ip set <ip>`	sets the IPv4 address of the WRPC(***)

* after executing _mode_ command, _ptp start_ is required to start WR PTP daemon in new mode

** requires running _sfp detect_ first

*** available only with Etherbone support compiled in