White Rabbit PTP Core User's Manual

December 2015 (wrpc-v3.0)
Building and Running

Grzegorz Daniluk (CERN BE-CO-HT)

Table of Contents

Introduction

This is the user manual for the White Rabbit PTP Core, part of the White Rabbit project. 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 please skip Chapter 2 [Building the Core], page 2 and move forward directly to Chapter 3 [Running and Configuring], page 4.

1 Software and hardware requirements

1.1 Repositories and Releases

This manual is about the official wrpc-v3.0 stable release of the White Rabbit PTP Core (WRPC).

The code and documentation for the project is distributed in the following places:

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

`http://www.ohwr.org/projects/wr-cores/files`
: place where you can find a synthesized bitstream, ready to be downloaded to SPEC, for every stable release

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

`git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git`
: read-only repository with the WRPC LM32 software

Other tools useful for building and running the WRPC can be downloaded from the following locations:

`git://ohwr.org/misc/hdl-make.git`
: *hdlmake* is used in the HDL synthesis process to create a 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 software

Repositories containing the WRPC gateware and software (*wr-cores*, *wrpc-sw*) are tagged with `wrpc-v3.0` tag. Other tools used to build the core and load it into SPEC board should be used in their newest stable releases.

1.2 Required hardware

The absolute minimum to run the WR PTP CORE is a PC computer with Linux and a Simple PCIe FMC Carrier (SPEC) - http://www.ohwr.org/projects/spec. However, it is highly recommended to use also the DIO FMC card (http://www.ohwr.org/projects/fmc-dio-5chttla) to be able to feed 1-PPS and 10MHz from external clock and output 1-PPS aligned to the WR time. To test the White Rabbit synchronization, you will also need:

- another SPEC board with a DIO FMC or a White Rabbit Switch;
- pair of WR-supported SFP transceivers (the list of supported SFPs can be found on our wiki page http://www.ohwr.org/projects/white-rabbit/wiki/SFP)
- a roll of G652, single mode fiber to connect your SPECs or SPEC with a WR Switch.

2 Building the Core

Note: you can skip this chapter if you want to use the release binaries available from *ohwr.org*.

Building the core is a two step process. First you have to synthesize the FPGA firmware (gateware) and then compile the software which will be executed by the LM32 soft-core processor. If you don't need to modify the LM32 software, you can skip the compilation stage since synthesized gateware already embeds the default software for the release.

2.1 HDL synthesis

Before running the synthesis process you have to make sure your environment is set up correctly. You need a Xilinx ISE software with at least a WebPack license. *ISE* provides a set of scripts: *settings32.sh*, *settings32.csh*, *settings64.sh* and *settings64.csh* that configure all the system variables required by the Xilinx software. Depending on a shell you use and whether your Linux is 32 or 64-bits you should execute one of them before the other tools are used. For 64-bit system and BASH shell you should call:

```
/opt/Xilinx/<version>/ISE_DS/settings64.sh
```

The easiest way to ensure that *ISE*-related variables are always set in your shell is adding the execution of the script to your *bash.rc* file. You can check if the shell is configured correctly by verifying if the *$XILINX* variable contains path to your *ISE* installation directory.

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

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

Note: the Xilinx project file included in the WRPC sources was created with Xilinx ISE 14.5. It is however recommended to use the newest available version of the ISE software.

HDL sources for the WR PTP CORE could be synthesized using Xilinx ISE without any additional tools, but using *hdlmake* is more convenient. It creates a synthesis Makefile and ISE project file based on a set of Manifest.py files deployed among the directories inside the *wr-cores* repository.

First, please clone the *hdlmake* repository from its location given in Section 1.1 [Repositories and Releases], page 1:

```
$ wget http://www.ohwr.org/attachments/download/2070/hdlmake-v1.0
$ git clone git://ohwr.org/misc/hdl-make.git <your_location>/hdl-make
$ cd <your_location>/hdl-make
$ git checkout 9d303ee
```

Then, using your favorite editor, you should create an *hdlmake* script in /usr/bin to be able to call it from any directory. The script should have the following content:

```
#!/usr/bin/env bash
python2.7 /path_to_hdlmake_sources/hdl-make/hdlmake/__main__.py $@
```

After that, you should make your script executable:

```
chmod a+x /usr/bin/hdlmake
```

Having Xilinx ISE software and *hdlmake* in place, 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*:

```
$ git clone git://ohwr.org/hdl-core-lib/wr-cores.git <your_location>/wr-cores
$ cd <your_location>/wr-cores
```

To build the gateware using sources of a stable release wrpc-v3.0, you have to checkout the proper git tag:

```
$ git checkout wrpc-v3.0
```

If you use *wr-cores* within another project (like *wr-nic*), you may need to check out another release tag for this repository. Please refer to the project's documentation to find out which version of this package you need to build.

You also need to fetch other git repositories containing modules instantiated inside the WR PTP CORE HDL. They are configured as git submodules:

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

The local copies of the submodules are stored to:

```
<your_location>/wr-cores/ip_cores
```

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

```
$ cd <your_location>/wr-cores/syn/spec_1_1/wr_core_demo/
```

First, please call *hdlmake* to create synthesis Makefile for Xilinx ISE:

```
$ hdlmake
```

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

```
$ make
```

This takes (depending on your computer speed) about 15 minutes and should create two files with FPGA firmware: *spec_top.bit* and *spec_top.bin*. The former can be downloaded to FPGA with Xilinx Platform Cable using e.g. *Xilinx Impact*. The latter can be used with kernel drivers from the *spec-sw* repository (check 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;

2.2 LM32 software compilation

Note: By default, the LM32 software for a stable release is embedded inside the FPGA bitstream you've downloaded from *ohwr.org* or synthesized in the previous step. This means you don't have to do a manual compilation of the LM32 software unless you want to use a development version or you've made some changes required by your application.

To compile the LM32 software for the White Rabbit PTP Core you will need to download and unpack the LM32 toolchain from the location mentioned in Section 1.1 [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>
```

Then you need to set a `CROSS_COMPILE` variable in order to compile the software for the LM32 processor:

```
$ export CROSS_COMPILE="<your_lm32_location>/lm32/bin/lm32-elf-"
```

To get the sources of the WRPC software please clone the *wrpc-sw* git repository tagged with wrpc-v3.0 tag. If you use WRPC within another project, you may need to checkout a different tag or a specific commit. If this applies, please refer to a documentation for this project.

```
$ git clone git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git <your_location>/wrpc-sw
$ cd <your_location>/wrpc-sw
$ git checkout wrpc-v3.0   # or "git checkout master"
```

Before you can compile *wrpc-sw* you need to make a few configuration choices. The package uses *Kconfig* as a configuration engine, so you may run one of the following commnads (the first is text-mode, the second uses a KDE GUI and the third uses a Gnome GUI):

```
$ make menuconfig
$ make xconfig
$ make gconfig
```

Other *Kconfig* target applies, like `config`, `oldconfig` and so on. A few default known-good configurations are found in `./configs` and you choose one by *make*ing it by name:

```
$ make spec_defconfig
```

The most important configuration choice at this point in time is whether to enable Etherbone or not. It is disabled by default in `spec_defconfig` and enabled by default in `etherbone_defconfig`.

After the package is configured, just run `make` without parameters to build your binary file:

```
$ make
```

The first time you build, the *Makefile* automatically downloads the *git submodules* of this package, unless you already did that by hand. The second and later build won't download anything from the network.

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

For this step you will need a SPEC board software support package (SPEC-SW) from *ohwr.org*. It is a set of Linux kernel drivers and userspace tools, that interact with a SPEC board plugged into PCI-Express slot.

Instructions in this section are based on a development version of SPEC-SW so if a stable release more recent than *2014-02* is available, you should use it instead.

If there is a more recent version of the SPEC software support, 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-SW package and build it:

```
$ git clone git://ohwr.org/fmc-projects/spec/spec-sw.git <your_specsw_location>
$ cd <your_specsw_location>
$ git checkout c0e18a7
$ make
```

Then you should copy your *spec_top.bin* generated in Section 2.1 [HDL synthesis], page 2 or downloaded from the *ohwr* to /lib/firmware/fmc/. changing its name:

Note: the commands below have to be executed with superuser rights

```
$ sudo cp <your_location>/wr-cores/syn/spec_1_1/wr_core_demo/spec_top.bin \
    /lib/firmware/fmc/spec-3.0.bin
```

You have to download also the "golden" firmware for SPEC card. It is used by the drivers to recognize the hardware:

```
$ wget http://www.ohwr.org/attachments/download/4057/spec-init.bin-2015-09-18
$ sudo mv spec-init.bin-2015-09-18 /lib/firmware/fmc/spec-init.bin
```

Now you can load the drivers necessary to access SPEC board from your system:

```
$ sudo insmod fmc-bus/kernel/fmc.ko
$ sudo insmod kernel/spec.ko
```

By default, when loading the *spec.ko* driver FPGA gets programmed with the "golden" bitstream. Starting from version 3.0, WR PTP CORE uses a flash memory chip on the carrier as a default place for storing the calibration parameters and the init script. Also the storage format of this information is now better organised in the files of the SDBFS filesystem. Therefore, starting from v3.0 you have to write the SDBFS filesystem image to the flash before running the WR PTP CORE. You can download the image from our project page:

```
$ wget http://www.ohwr.org/attachments/download/4060/sdbfs-flash.bin
```

It contains all the files required by the WR PTP CORE. They are empty, but have to exist in the SDBFS structure to be written later as described in Section 3.2 [Writing configuration], page 6. To store the filesystem image in flash, please execute the following command:

```
$ sudo tools/flash-write -b 0x20 -c 0x0 0 1507712 < \
    <your_location>/sdbfs-flash.bin
```

Note: Please refer to Appendix C [Writing SDBFS image in standalone configuration], page 14 for instructions on how to write the SDBFS image to a standalone SPEC or custom hardware.

Now, you are ready to load the last driver, which downloads the actual WR PTP CORE bitstream to the Spartan 6 FPGA:

```
$ sudo insmod fmc-bus/kernel/fmc-trivial.ko gateware=fmc/spec-3.0.bin
```

You can use the *dmesg* Linux command to verify if the FPGA firmware file was loaded into the FPGA. Among plenty of messages you should be able to find something very similar to:

```
[1275526.738895] spec 0000:20:00.0:  probe for device 0020:0000
[1275526.738906] spec 0000:20:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16
[1275526.738913] spec 0000:20:00.0: setting latency timer to 64
[1275526.743102] spec 0000:20:00.0: got file "fmc/spec-init.bin", 1485236 (0x16a9b4) bytes
[1275526.934710] spec 0000:20:00.0: FPGA programming successful
[1275527.296754] spec 0000:20:00.0: mezzanine 0
[1275527.296756]       Manufacturer: CERN
[1275527.296757]       Product name: FmcDio5cha

[1275593.973147] fmc FmcDio5cha-2000: Driver has no ID: matches all
[1275593.973177] spec 0000:20:00.0: reprogramming with fmc/spec-3.0.bin
[1275594.168249] spec 0000:20:00.0: FPGA programming successful
```

If everything went right up to this moment you have your board running the FPGA bitstream with a default LM32 software. If you want to load your own *wrc.bin* built from the *wrpc-sw* repository you can use the *spec-cl* tool:

```
$ sudo tools/spec-cl <your_location>/wrpc-sw/wrc.bin
```

Now you should be able to start a Virtual-UART tool (also part of the SPEC-SW package) that will be used to interact with the WR PTP CORE shell:

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

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

3.2 Writing configuration

First, you should perform a few configuration steps through the WRPC shell before using the core.

Note: the examples below describe only a subset of the WRPC Shell commands. The full list of supported commands can be found in Appendix A [WRPC Shell commands], page 10.

Before making the configuration changes, it is good to stop the PTP daemon. Then, debug messages from the daemon will not show up to the console while you interact with the shell.

```
wrc# ptp stop
```

First you should make sure your board has a proper MAC address assigned:

```
wrc# mac get
```

If the result of above command is `MAC-address: 22:33:ww:xx:yy:zz`, this means MAC was not yet configured and stored in the Flash/EEPROM. The value is based on thermometer serial number as is unique among SPEC devices, globally accepted as "locally assigned", but you might want to assign your own address. A value `22:33:44:55:66:77` is the final fallback if no thermometer is found (very unlikely). You should get the MAC for your board from its manufacturer. To configure the address and store it into the Flash/EEPROM (so that it's automatically loaded every time the WRPC starts) you should type two commands in the WRPC shell:

```
wrc# mac set xx:xx:xx:xx:xx:xx
wrc# mac setp xx:xx:xx:xx:xx:xx
```

where `xx:xx:xx:xx:xx:xx` is the MAC address of your board.

Next you should create a calibration database with fixed delays values and alpha parameters. The example below presents the WRPC Shell commands that clear all previous entries and add two Axcen transceivers with deltaTx, deltaRx and alpha parameters associated with them.

```
wrc# sfp erase
wrc# sfp add AXGE-1254-0531 180625 148451 72169888
wrc# sfp add AXGE-3454-0531 180625 148451 -73685416
```

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

Note: The deltaTx and deltaRx parameters above are the defaults for wrpc-v3.0 release bitstream available on *ohwr.org*, running on SPEC v4 board and calibrated to port 1 of a WR Switch v3.3. These values as well as the parameters for the WR Switch are available on the calibration wiki page (*http://www.ohwr.org/projects/white-rabbit/wiki/Calibration*). However, if you re-synthesize the firmware or want to have the most accurate estimation of the fixed delays and alpha for your fiber, you should read and perform the WR Calibration procedure (*http://www.ohwr.org/documents/213*).

The WR PTP CORE mode of operation (GrandMaster/Master/Slave) can be set using the *mode* command:

```
wrc# mode gm       # for GrandMaster mode
wrc# mode master   # for Master mode
wrc# mode slave    # for Slave mode
```

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

```
wrc# ptp start
```

Note: For running the GrandMaster mode, you need to provide 1-PPS and 10MHz signal from an external source (e.g. GPS receiver or Cesium clock). Please connect 1-PPS signal to the LEMO connector No.4 and 10MHz to the LEMO connector No.5 on the FMC DIO mezzanine board.

One option is to type all the commands to initialize the WRPC software to the required state every time the Core starts. However, you can also write your own initialization script to the Flash/EEPROM. It will be executed every time the WRPC software starts. A simple script that loads the calibration parameters, configures the WR mode to Slave and starts the PTP daemon is presented below:

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

Almost exactly the same one can be used for running WRPC in the GrandMaster or Master mode. The only difference would be changing the *init add mode slave* line to *init add mode gm* or *init add mode master*.

3.3 Running the Core

Having the SFP database, and the init script created in Section 3.2 [Writing configuration], page 6 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
```

You should see log messages that confirm the execution of the initialization script:

```
executing: ptp stop
executing: sfp detect
AXGE-3454-0531
executing: sfp match
SFP matched, dTx=180707, dRx=148323, alpha=-73685416
executing: mode slave
Locking PLL
executing: ptp start
Slave Only, clock class set to 255
```

Now you should have the WR PTP CORE running in WR Slave mode. WRPC needs to make a calibration of t24p phase transition value. It has to be done only once for a new bitstream and is performed automatically when WRPC runs in the Slave mode. That is why it is very important, even if WRPC is meant to run in the Master mode, to configure it to Slave for a moment and connect to any WR Master. This has to be repeated every time a new bitstream (gateware) is deployed. The measured value is automatically stored to Flash/EEPROM and used later in the Master or GrandMaster mode.

The Shell also contains a monitoring function which you can use to check the WR synchronization status:

```
wrc# gui
```

The information is presented in a clear, auto-refreshing screen. The information is refreshed at every WR iteration or periodically if nothing else happens (so you see an up-to-date timestamp). The period defaults to 1 second and can be changed using the *refresh* command. To exit from

this console mode press <Esc>. A full description of the information reported by gui is provided in Appendix B [WRPC GUI elements], page 13.

Note: the *Synchronization status* and *Timing parameters* in *gui* are available only in the 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.

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

TAI Time:                 Fri, Jan 2, 1970, 03:12:46

wru1: Link up   (RX: 742, TX: 213), mode: WR Slave   Locked  Calibrated

PTP status: slave

Synchronization status:

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

Timing parameters:

Round-trip time (mu):    833913 ps
Master-slave delay:      398192 ps
Master PHY delays:       TX: 10 ps, RX: 163610 ps
Slave PHY delays:        TX: 0 ps, RX: 126000 ps
Total link asymmetry:       37529 ps
Cable rtt delay:            78680 ps
Clock offset:                  -2 ps
Phase setpoint:             5341 ps
Skew:                          -2 ps
Manual phase adjustment:        0 ps
Update counter:               184
```

If you want to log statistics from the WRPC operation, it's probably better to use the *stat* shell command. It reports the same information as GUI but in a single long line, a form which is easier to parse and analyze:

```
wrc# stat
lnk:1 rx:416 tx:118 lock:1 sv:1 ss:'TRACK_PHASE' aux:0 sec:94197 \
nsec:793068184 mu:836241 dms:400556 dtxm:10 drxm:163610 dtxs:0 drxs:128400 \
asym:35129 crtt:544221 cko:-5 setp:7667 hd:61479 md:37221 ad:65000 ucnt:101 \
temp: 45.6875 C
lnk:1 rx:417 tx:119 lock:1 sv:1 ss:'TRACK_PHASE' aux:0 sec:94198 \
nsec:293076296 mu:836253 dms:400562 dtxm:10 drxm:163610 dtxs:0 drxs:128400 \
asym:35129 crtt:544233 cko:-4 setp:7663 hd:61485 md:37259 ad:65000 ucnt:102 \
temp: 45.6875 C
(...)
```

Unlike *gui*, the *stat* command runs asynchronously: you can still issue shell commands while stats are running (this is different from earlier `wrpc-sw` releases). You can stop statistics by running *stat* again. As an alternative to the toggling action of *stat* alone, you can use "*stat 1*" or "*stat 0*".

Statistics are printed every time the WR servo runs; thus no statistics are reported when the node is running in master mode, nor when your node is running as slave and the master disappeared.

If you have a DIO mezzanine board plugged to your SPEC, you can verify the synchronization performance by observing the offset between 1-PPS signals from the WR Master and WR Slave. The WR PTP CORE generates 1-PPS signal on the LEMO connector No. 1. Please remember to use oscilloscope cables of 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 or fmc-trivial.ko driver.

This will occur when you try to load the driver while your *spec-vuart* is running and trying to get messages from Virtual-UART's registers inside the 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 (*settings64.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.

If you're trying to synchronize the SPEC board to WR Switch please remember to read the document and perform the WR Calibration to find out the values of deltaRx and deltaTx for your firmware. Check if the oscilloscope cables you use have the same delays (or take the delay difference into account in your measurements).

5 Questions, reporting bugs

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

Appendix A WRPC Shell Commands

`help`	lists available commands in this instance of the WRPC
`ver`	prints which version of wrpc is running
`config`	prints the Kconfig file used to build this instance of WRPC. It is an optional command, enabled at build time by `CONFIG_CMD_CONFIG`
`verbose <digits>`	sets PPSi verbosity. See the PPSi manual about the meaning of the digits (hint: `verbose 1111` is a good first bet to see how the PTP system is working)
`pll init <mode> <ref_channel> <align_pps>`	manually runs spll_init() function to initialize SoftPll
`pll cl <channel>`	checks if SoftPLL is locked for the channel
`pll sps <channel> <picoseconds>`	sets phase shift for the channel
`pll gps <channel>`	gets current and target phase shift for the channel
`pll start <channel>`	starts SoftPLL for the channel
`pll stop <channel>`	stops SoftPLL for the channel
`pll sdac <index> <val>`	sets the dac
`pll gdac <index>`	gets dac's value
`gui`	starts GUI WRPC monitor
`stat`	toggles reporting of loggable statistics. You can pass `1` or `0` as argument as an alternative to toggling
`stat bts`	prints bitslide value for established WR Link, needed by the calibration procedure
`refresh`	changes the update time period of the gui and stat commands. Default period is 1 second. If you set the period to 0, the log message is only generated one time.
`ptp start`	starts WR PTP daemon
`ptp stop`	stops WR PTP daemon
`mode`	prints the current WR PTP mode
`mode gm|master|slave`	sets WRPC to operate as Grandmaster clock (requires external 10MHz and 1-PPS reference), Master or Slave. After setting the mode, `ptp start` must be re-issued

`calibration`	tries to read t2/4 phase transition value from the Flash/EEPROM (in WR Master or GrandMaster mode), or executes the t24p calibration procedure and stores its result to the Flash/EEPROM (in WR Slave mode)
`time`	prints current time from WRPC
`time raw`	prints current time in a raw format (seconds, nanoseconds)
`time set <sec> <nsec>`	sets WRPC time
`time setsec <sec>`	sets only seconds of the WRPC time (useful for setting time in GrandMaster mode, when nanoseconds counter is aligned to external 1-PPS and 10 MHz)
`time setnsec <nsec>`	sets only nanoseconds of the WRPC time
`sfp detect`	prints the ID of a currently used SFP transceiver
`sfp erase`	erases the SFP database stored in the Flash/EEPROM
`sfp add <ID> <deltaTx> <deltaRx> <alpha>`	stores calibration parameters for SFP to a file in Flash/EEPROM
`sfp show`	prints all SFP transceivers stored in database
`sfp match`	tries to load the calibration parameters for currently used SFP transceiver (`sfp detect` must be executed before `match`)
`init erase`	erases the initialization script in Flash/EEPROM
`init add <cmd>`	adds shell command at the end of the initialization script
`init show`	prints all commands from the script stored in Flash/EEPROM
`init boot`	executes the script stored in Flash/EEPROM (the same action is done automatically when WRPC starts after resetting LM32)
`mac get`	prints WRPC's MAC address
`mac getp`	reads the MAC address stored in Flash/EEPROM
`mac set <mac>`	sets the MAC address of WRPC
`mac setp <mac>`	stores the MAC address in Flash/EEPROM
`sdb`	prints devices connected to the Wishbone bus inside WRPC

`ip get` `ip set <ip>`	reports or sets the IPv4 address of the WRPC (only available if `CONFIG_ETHERBONE` is set at build time
`w1w <offset> <byte> [<byte> ...]` `w1r <offset> <len>`	If `CONFIG_W1` is set and a OneWire EEPROM exists, write and read data. For writing, `byte` values are decimal

Appendix B WRPC GUI elements

`TAI Time:`	current state of device's local clock
`RX: / TX:`	Rx/Tx packets counters
`mode:`	operation mode of the WR PTP CORE - `<WR Master, WR Slave>`
`< Locked, NoLock >`	SoftPLL lock state
`Servo state:`	current state of WR servo state machine - `<Uninitialized, SYNC_SEC, SYNC_NSEC, SYNC_PHASE, TRACK_PHASE>`
`Phase tracking:`	is phase tracking enabled when WR Slave is synchronized to WR Master - `<ON, OFF>`
`Synchronization source:`	network interface name from which WR daemon gets synchronization - `<wru1>`
`Round-trip time (mu):`	round-trip delay in picoseconds ($delay_{MM}$)
`Master-slave delay:`	estimated one-way (master to slave) link delay ($delay_{MS}$)
`Master PHY delays:`	transmission/reception delays of WR Master's hardware ($\Delta_{TXM}, \Delta_{RXM}$)
`Slave PHY delays:`	transmission/reception delays of WR Slave's hardware ($\Delta_{TXS}, \Delta_{RXS}$)
`Total link asymmetry:`	WR link asymmetry calculated as $delay_{MM} - 2 \cdot delay_{MS}$
`Cable rtt delay:`	round-trip fiber latency
`Clock offset:`	Slave to Master offset calculated by PTP daemon ($offset_{MS}$)
`Phase setpoint:`	current Slave's clock phase shift value
`Skew:`	the difference between current and previous estimated one-way link delay
`Update counter:`	the state of counter incremented every time the WR servo is updated

Appendix C Writing SDBFS image in standalone configuration

If you use SPEC board in a host-less environment, or you use custom hardware and SPEC drivers and tools cannot be used, there is still a possibility of writing SDBFS through Xilinx JTAG.

In case of SPEC running a reference bitstream provided with a stable WRPC release, you can simply program your Flash with *spec_top.mcs* provided with the release binaries using for example Xilinx ISE Impact tool. This *mcs* file already includes both SDBFS image and FPGA bitstream.

In case of a custom gateware or hardware, you can download a standalone SDBFS image:

```
$ wget http://www.ohwr.org/attachments/download/4144/sdbfs-standalone.bin
```

and generate a custom **.mcs* file with your own FPGA bitstream. You should use the following layout:

`0x000000` your FPGA bitstream
`0x170000` SDBFS image

For example, to generate the **.mcs* file for M25P32 Flash on SPEC, the following *promgen* parameters should be used:

```
promgen -w -spi -p mcs -c FF -s 32768 -u 0 <your_bitstream>.bit \
-bd sdb-standalone.bin start 0x170000 -o output.mcs
```

After that, you can use the Xilinx JTAG cable and Xilinx ISE Impact tool to write your *output.mcs* file to the Flash memory.