SPEC Demonstration Package: User Manual

September 2011

Document edited for CERN by Alessandro Rubini.

Table of Contents

Introduction

This document is the user manual for the demonstration package for the SPEC board. It explains how to install and build all the software needed to experience *White Rabbit* in your own lab.

Please note that the description here goes to some detail about rebuilding the software, but most of it can be skipped in a first reading, if you just need to run the demo without making you hands dirty.

1 Description of the Package

White Rabbit is a synchronization system designed as an extension of PTP (*Precision Time Protocol*). It is able to achieve sub-nanosecond accuracy over a multi-level tree of nodes.

SPEC, the *Simple PCI-Express Carrier* is a carrier for FMC cards; it is able to run the *White Rabbit* engine to drive the I/O mezzanine cards with accurate time stamps. The SPEC included in the demo package is equipped with a 5-channel digital I/O card.

The *White Rabbit Switch*, part of the demo package, is a Gigabit switch for fiber links using SFP transceivers, running White Rabbit as either a master or a slave. In the demo package it is used as a master, with the SPEC acting as a slave.

2 Ethernet Wiring

To achieve synchronization of the two systems, the asymmetry of signalling must be taken care of. The SPEC and switch should be connected with a single-mode fiber. The software shipped for the demo package includes configuration settings for this configuration:

- The **violet** SFP transceiver must be plugged in the `UP0` or `UP1` ports of the switch. Fine calibration is not yet supported on the other ports.
- The **blue** SFP transceiver must be plugged in the SPEC card (which in turn is plugged in a 4-lane PCI-E slot of a GNU/Linux host).

If you swap the transceivers nothing dramatic is going to happen, but the two nodes won't synchronize correctly (the error depends on the length of your fiber).

3 Booting The Switch

The *White Rabbit Switch* runs a GNU/Linux system in the control processor, while the WR core is implemented in FPGA. The control processor loads the FPGA and runs the WR-PTP protocol.

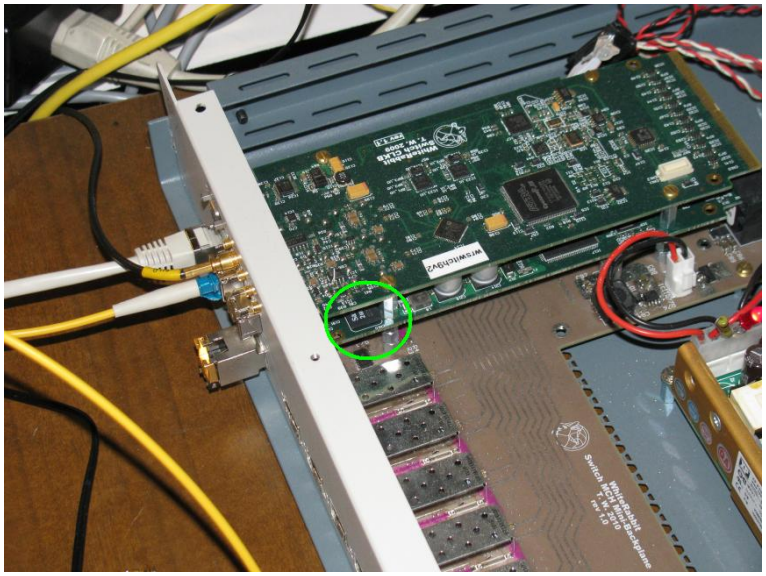
3.1 Booting Options

To boot the switch, you can use either *bootp* and *tftp* or the micro-SD card. While network boot is preferred during development, we suggest using the SD card to run the demo (the switch is already shipped with a card inside).

3.2 Using the Shipped SD card

When you received the kit, there was a micro-SD card in the switch. You'll most likely want to update it with the archive called `wrsw-filesystem.tar.gz`. The archive is a 4.5MB tar file and can be downloaded from the *files* section of the *White Rabbit Switch Software* OHWR project. The direct link while this is edited is http://www.ohwr.org/attachments/download/717/wrsw-filesystem.tar.gz, but you may find an updated version in the *files* section of the OHWR project. [**Note:** This isn't the same binary you used with previous versions of this manual].

You should remove the top cover of the switch; you'll need to remove 4 screws, and you can even leave it open during operation. The micro-SD card is plugged in a slot within the switch PCB: it is marked with a green circle in the photograph below:



The card has two partitions, you should mount the second partition (which is formatted with *ext2*) and overwrite it with the *tar* file. If your host system auto-mounted the partition, you need to skip the first command and figure what is the mount point that has been created (to be used in place of `/mnt`).

```
mount /dev/sdb2 /mnt
tar -C /mnt -xvzf wrsw-filesystem.tar.gz
umount /mnt
```

Finally, put the card back in the switch and power it up.

If you just need to run the demo (as opposed to be more involved in development) you may jump over the rest of the chapter and go to Chapter 4 [Running the SPEC], page 4.

3.3 Using a New SD Card

If you have the switch and want to use your own card, you need to create two partitions in it; the first being a thin *fat* and the second being *ext2*. The reason for this is that the internal ROM of the control CPU can only boot from a *fat* partition, which is however unusable to host a real Unix system.

Unfortunately, the ROM is quite picky about the layout of the *fat* filesystem; creating the right filesystem is mainly a matter of luck, so Tomasz Wlostowski published an already made image of the whole card. Here are his instructions:

- Download http://svn.ohwr.org/white-rabbit/trunk/hdl/bin/sdimage.bz2;

- Take a microSD card up to 2 GB (it must be a micro-SD card, SDHC cards won't work);
- Write the image onto the card (as root): `bzcat sdimage2.bz2 > /dev/YOUR_SD_CARD_DEVICE`. The device is usually `/dev/mmcblkX` or `/dev/sdX`, for some `X`. Note that this is an image of an entire card (not a single partition).
- Be careful and don't accidentally erase your hard drive.
- Update the card with `wrsw-filesystem.tar.gz` (see Section 3.2 [Using the Shipped SD card], page 2).
- Insert the card into the slot on the main board of the switch.

This tricky stuff "just works" and is not going to be cleaned up, as V3 of the WR Switch won't use the SD card nor even the same CPU. Anyway, if you are interested, complete sources are available at `http://twlostow.web.cern.ch/wr/boot-sd.tar.bz2` (please note it's 32MB, as the procedure involves a complete *barebox* boot loader, so the sources of it all are included).

3.4 Network Boot

If you are involved in *White Rabbit* you'll surely be interested in network boot of the switch. When no SD card is found, the CPU ROM boots from an internal flash where a boot loader has been installed during production. So you can just remove the SD card and the system will use the usual *bootp/tftp* sequence; please note that you need to have the respective servers installed and configured in your network.

The network cable being used for boot is the 100Mb interface of the control CPU, available as RJ45 on the front panel of the switch (you can see a white UTP cable plugged to it in the previous photograph).

The boot loader will download from the net the following files:

- `uImage` (the kernel image, encapsulated for *u-boot*);
- `ramdisk.ext2` (the filesystem; either a compressed *ext2* image or an *initramfs* – the latter is preferred)

To run the demo, you'll most likely want to use the same `wrsw-filesystem.tar.gz` that you would place in the internal SD card. To turn the *tar* file into an *initramfs*, run the following commands as root:

```
mkdir untar-fs
cd untar-fs
tar -xzf ../wrsw-filesystem.tar.gz
find . -print | cpio -o -H newc | gzip > ../ramdisk.ext2
```

As explained, *ramdisk.ext2* is not an *ext2* filesystem, but it's the name that the boot-loader has been requesting since the initial port.

When booting from the network you'll most likely want to have a serial console connected to the device, as described next.

3.5 Connecting to the Switch

If you are going to develop with the switch, you'll need to interact with its command line. Two channels are available: the serial console (for which you need a flat cable and a level converter) and the 100Mb network interface.

The control CPU runs an *ssh* server, where you can login as `root` with an empty password – at least this is the configuration in the demo and development setup.

Please note that the switch asks for an IP addres through DHCP, so the address being shown in the example below belongs to my local LAN address range; missing a serial port, you can

retrieve the IP address from the log files in your DHCP server. It's worth noting that V3 of the switch will have an easily accessible UART port.

As usual, you'll need to manually confirm you trust the remote system when you connect the first time:

```
rudo% ssh root@192.168.16.224
The authenticity of host '192.168.16.224 (192.168.16.224)'
          can't be established.
RSA key fingerprint is b3:19:b8:29:d9:0e:53:4c:c0:55:97:19:ab:2f:92:cc.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.16.224' (RSA) to the list of known hosts.
root@192.168.16.224's password:
# ifconfig | grep wr
wrd0      Link encap:Ethernet  HWaddr 02:4A:BC:00:00:00
wrd1      Link encap:Ethernet  HWaddr 02:4A:BC:00:00:01
wru0      Link encap:Ethernet  HWaddr 02:4A:BC:00:00:20
wru1      Link encap:Ethernet  HWaddr 02:4A:BC:00:00:21
```

As you see, the system has configured 4 *White Rabbit* ports – but remember to plug your fiber to one of the *uplink* ports, as explained in Chapter 2 [Ethernet Wiring], page 1.

If you have a serial console, you'll see something like the following during a successful system boot:

```
U-Boot 1.3.4 (Feb 23 2010 - 16:37:22)

DRAM:  64 MB
NAND:  No NAND device found!!!
0 MiB
DataFlash:AT45DB642
[...]
TFTP from server 192.168.16.1; our IP address is 192.168.16.245
Filename 'uImage'.
Load address: 0x22000000
Loading: ###################################################################
[...]
Filename 'ramdisk.ext2'.
Load address: 0x23000000
Loading: ###################################################################
[...]
Linux version 2.6.35 (gcc version 4.4.2 (GCC) ) #1
[...]
Starting dropbear sshd: OK
Starting WR Hardware Abstraction Layer daemon:
[...]
(I) [hal_ports.c:334]    Initializing switch ports
(I) [hal_ports.c:254]    Initializing port 'wru1' [02:4a:bc:00:00:21]
(I) [hal_ports.c:283]    Alpha: 0.0002733400 FixAlpha: 75124859
(I) [hal_ports.c:318]    Port wru1: mode wr_m_and_s
(I) [hal_ports.c:254]    Initializing port 'wru0' [02:4a:bc:00:00:20]
(I) [hal_ports.c:283]    Alpha: 0.0002733400 FixAlpha: 75124859
(I) [hal_ports.c:318]    Port wru0: mode wr_m_and_s
[...]
WRPTP.v2 daemon started. HAVE FUN !!!
```

4 Running the SPEC

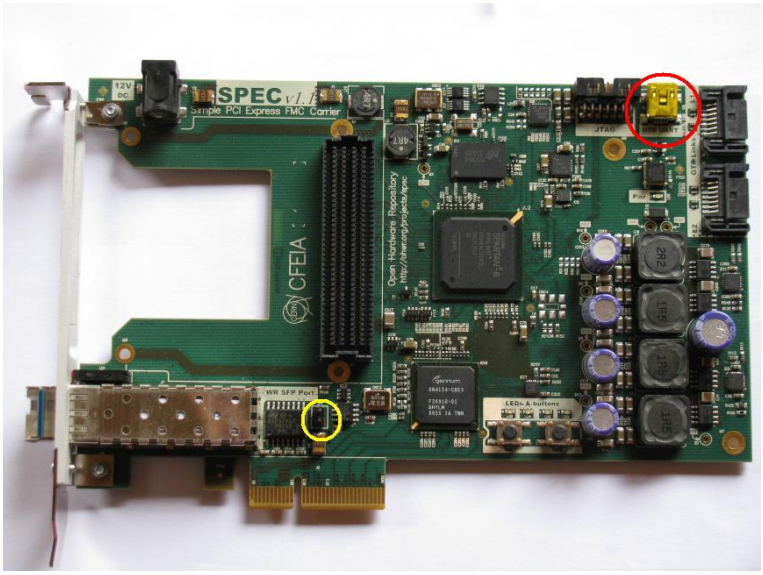
The SPEC card must be plugged in a 4-lane or larger PCI-Express slot, in a GNU/Linux host. It will appear in *lspci* as follows:

```
03:00.0 Non-VGA unclassified device: Device 1a39:0004 (rev 01)
```

The exact bus number (here "03" will be different on your host, but that's irrelevant.

To run the WR software suite, you must perform two steps: loading a Linux driver in the host OS and then loading the FPGA bitstream through the driver. Such a bitstream includes an LM32 soft-core which runs its own embedded WRPTPD executable, which spits diagnostic messages to a serial port.

The following image shows the card. The red circle identifies the USB connector (the serial port for diagnostics); the yellow circle shows the jumper I'll refer to in a while.



4.1 The Kernel Driver

The device driver you need is for the *Gennum GN4124* PCIe-to-localbus bridge. The GN name suggested the *gnurabbit* name for the package.

Unfortunately, no binary can be shipped for this driver as you need to recompile it for the specific kernel version that you are running.

The package, with documentation, can be downloaded with *git* from `git://gnudd.com/gnurabbit.git` or `git://github.com/a-rubini/gnurabbit.git`, or as a *tar* file from `http://www.ohwr.org/attachments/download/16/gnurabbit-2011-09-01.tar.gz`.

Follow the instructions to compile it (subdirectory `doc`), which can be summarized as:

```
export LINUX=/path/to/your-kernel
make
```

(Please ignore any errors about documentation: the package relies on TEX and other packages being available on the host system, but you may well not have installed them; also there's a known warning in the code about an unused function, it's not an issue at all for you – it is for me).

Load the kernel module as superuser. The module is called `spec-demo.ko` as it is specifically designed for the SPEC, but the final SPEC driver will be different. Previous versions of this document referred to the generic `rawrabbit.ko` driver; the old procedure is still working but this driver is easier to use.

```
insmod kernel/spec-demo.ko
```

Actually, before loading the module, you may want to place the two binary files for the SPEC in */lib/firmware*. You'll need `spec_top.bin` and `wrc.bin`. Both can be downloaded from Tom's personal space:

```
http://twlostow.web.cern.ch/twlostow/wr-firmware-20110824/
```

A copy of those binaries is available in the *files* section of the *spec-sw* project in OHWR:

```
http://www.ohwr.org/projects/spec-sw/files
```

As of this writing, the direct links are as follows, but please note that newer files may be there by the time you read this:

```
http://www.ohwr.org/attachments/download/718/spec_top.bin
http://www.ohwr.org/attachments/download/719/wrc.bin
```

With the two files under */lib/firmware*, you can load the module. Your kernel messages (on the host) will look like the following:

```
[22300.886172] request firmware "spec_top.bin": 0 (Success)
[22300.892432] rr_loader_complete: got firmware file, 1484404 (0x16a674) bytes
[22301.090518] __rr_gennum_load: done after 371101 writes
[22301.095990] request program "wrc.bin": 0 (Success)
[22301.099923] spec_loader_complete: got program file, 49044 (0xbf94) bytes
[22301.101416] LM32 has been restarted
```

Note: while */lib/firmware* is most likely the correct directory to place your firmware in, the actual name on your host system may be different according to how your GNU/Linux distribution is configured.

More information about the kernel module is in the manual for *gnurabbit*, at `git://gnudd.com/gnurabbit.git` or `git://github.com/a-rubini/gnurabbit.git`. The current version is available both as source archive and precompiled PDF for the manual at *ohwr.org*:

```
http://www.ohwr.org/attachments/715/gnurabbit-2011-09-01.pdf
http://www.ohwr.org/attachments/download/716/gnurabbit-2011-09-01.tar.gz
```

4.2 The FPGA Binary

The FPGA binary bitstream for the SPEC is loaded directly by the kernel module. If you recompile the VHDL code you'll need to place the new binary in */lib/firmware* and re-load the module:

```
cd /path/to/gnurabbit
rmmod spec-demo
insmod kernel/spec-demo.ko
```

4.3 Diagnostic Messages

The CPU inside the spec runs an embedded WRPTPD, which spits diagnostic messages to its own serial port. The SPEC includes a serial-usb converter, so you can plug a mini-usb cable to a USB port of the same computer.

You'll need the `cp210x` driver in your host system; this is normally shipped as an auto-loaded kernel module by all distributions, but if recompiled your own kernel you'll need to add it to the configuration. The configuration symbol is `CONFIG_USB_SERIAL_CP210X`, enabled by `CONFIG_USB_SERIAL`.

You can run a communication program at the usual 115200,8N1 configuration and see diagnostic messages. Please note that no message appears before the link on the fiber is detected.

A normal run will show the following messages:

```
wr_core: starting up (press G to launch the GUI and D for extra debug messages).
...
Deltas: txm 20784 rxm 13410 txs 0 rxs 231200
->t1 = 1285702848:777016328:0
->t2 = 106:28276976:1642
->t3 = 106:214910952:0
->t4 = 1285702848:963758728:2614
->mdelay = 0:0:0
->HWOffset = 1285702742:748793680:1263
[...]
```

```
->t1 = 1285702877:396933896:0
->t2 = 1285702877:396988224:2915
->t3 = 1285702877:647817304:0
->t4 = 1285702877:647871400:1367
->mdelay = 0:108425:3276
->HWOffset = 0:0:3
```

The readers with a little exposure with PTP will identify in the abstract above the 4 time stamps used to synchronize the clocks.

All times here are in the form "*seconds*`:`*nanoseconds*`:`*picoseconds*" where *nanoseconds* is a multiple of 8ns (i.e., an integer count of the 125MHz clock) and *picoseconds* is in the range 0-8000 (with some jitter to nearby values outside the interval).

The 6 lines from `t1=` to `HWOffset` are repeated every second. As you see above, the first instance referred the the system right after boot, not synchronized, while the next (after the dots) shows a synchnonized system.

This console interface reads the UART as well: as the first line claims, if you type `d` you'll get more information and if you type `g` you'll get a "GUI" (made with escape sequences to have a full-screen ASCII status window).

The "GUI" status window is like the following one (please note that it doesn't fit into the standard 80x25 terminal, you'll need to enlarge your *minicom* terminal window):

```
WR PTP Core Sync Monitor v 0.2
g = exit, t = enable/disable phase tracking, +/- = increase/decrease phase by 100ps

Link status:

wru1: Link up   (RX: 5476, TX: 1730), mode: WR Slave   Locked  Calibrated

Synchronization status:

Servo state:               TRACK_PHASE
Phase tracking:            ON
Synchronization source:    wru1

Timing parameters:

Round-trip time (mu):      108428305 ps
Master-slave delay:        54330929 ps
Master PHY delays:         TX: 20784 ps, RX: 13410 ps
Slave PHY delays:          TX: 0 ps, RX: 231200 ps
Total link asymmetry:      -233553 ps
Clock offset:              -6 ps
Phase setpoint:            1281 ps
Skew:                      10 ps
Manual phase adjustment:   0 ps
```

5 Demo Results

After the two WRPTPD programs (one running on the switch and one running on the SPEC card) synchronize, you can measure the offset the respective clocks signals with a scope.

On the switch, you should plug a cable to the output labelled "PPS" on the metal case. This is the "pulse per second" signal, a positive pulse lasting 800us at the beginning of each TAI (or UTC) second – in this case we are using no external master clock, so it will be a local reference disconnected from reality.[1]

[1] The careful reader will have noticed that the UART messages shown refer to September 28th 2010, when the SPEC didn't yet exist.

On the SPEC, a similar PPS signal is output on the first digital output channel, which you can look at by plugging a cable. The photograph above shows the SPEC with the optical fiber (yellow) and PPS cable (brown).

Your scope, then, will show that the two PPS are well aligned, thanks to WRPTPD. The following image shows a scope screen-short with a horizontale scale of 200ns per square.

6 Troubleshooting

As in every demo, something may go wrong. These are some suggestions that may (or may not) help finding and fixing the issue you are facing.

6.1 Check the Fiber and Transceivers

If the SPEC gives no sign of life, there have been issues in programming the FPGA. Older versions of `wrc.bin` didn't report anything on the serial port before the optical link went up, but this is no longer the case.

If the optical link is down, you'll see no green led on the SPEC (the red led is always on to mean "I'm alive", the green led is used to report the link status. Also, the GUI will say "`wru1: Link down`". This may happen because your your fiber is too short, the link is not established if the optical transceivers are saturated. In this case you may need to get a longe fiber or make a few knots (1cm in diameter: not less or the fiber may break) to increase attenuation.

6.2 Replace the LM32 Program

If the FPGA is properly programmed, you can replace the LM32 program without doing complete FPGA reprogramming. You can reload `spec-demo.ko` with the parameter `fwname=none`, so only the `wrc.bin` is loaded, without a complete reprogramming of the FPGA. As an alternative, you can use the user-space loader part of *gnurabbit*.

For more details on both options, see the *gnurabbit* manual.

6.3 Stick to Precompiled Binaries

While all of this is free software and you can rebuild everything from sources (see Chapter 8 [Rebuilding from Source Files], page 9 for a complete list of related repositories), there may be some minor but details that are not immediately grasped.

On the other hand, it may happen that the demo is not compiled from the *master* branch of the repository – at time of writing, this happens for the *ptp* code, see Chapter 8 [Rebuilding from Source Files], page 9.

Therefore, before you try running your recompiled binaries, whether on the switch or on the SPEC, you'd better ensure the shipped binaries work at your site. If they do, then we can help in sorting out what is going differently in your recompiled binaries.

Please remember this is a demo and the project is still under development; moreover, this is vacation time in most of Europe so thinkgs won't change much in the next month.

7 Summary of Sofware and Hardware

This is the complete list of data and material you need to run the demo in your lab.

- The White Rabbit Switch, version 2.
- A micro-SD card, not from the high-capacity family.
- The filesystem for the switch, to be put in the micro-SD or TFTP.
- A roll of fiber to connect the switch and the SPEC. This fiber should be long enough so that its attenuation guarantees that the optical receiver is not overdriven by the optical emitter. The shipped SFP modules are relatively powerful because they are meant to drive 10 km links. For more information check the specifications on the web.
- Two optical SFP transceivers: 1310nm (blue) and 1490nm (violet).
- The SPEC with its digital I/O mezzanine board.
- A PCI-E PC running a recent kernel version (e.g. 2.6.35, 2.6.39).
- The compiler, module utils and *minicom* from your distribution.
- The *gnurabbit* device driver and
- the `spec_top.bin` binary file.
- An USB to mini-USB cable to see diagnostic messages.
- A coax cable to pick the PPS-OUT from the switch (SMC connector).
- A coax cable to pick the PPS-OUT from the SPEC I/O mezzanine (LEMO connector).
- A scope.

8 Rebuilding from Source Files

The software being used in the demo is all available as source code, with a Free Software license. Currently, the various components are split in several different repositories, as they are being developed by different groups of people, the situation is still somewhat in flux toward a more consolidated setup.

The relevant code and documentation is available from the following repositories, but please check for newer versions of this document at `http://www.ohwr.org/projects/spec-sw/files/` before starting a rebuild effort.

- WRPTPD is currently hosted at `git://gnudd.com/ptp-noposix.git`.
- The software for the switch is at `git://ohwr.org/white-rabbit/wr-switch-sw.git`. The package refers to WRPTPD during compilation; please note that the demo uses the branch called *ptpx-to-merge* (which isn't really ready to be merged, so it isn't part of *master* yet).
- The SD boot temporary solution is at `http://twlostow.web.cern.ch/wr/`.
- The lm32 code base is at `git://github.com/twlostow/wr-core-software.git`. This refers to WRPTPD during compilation (again, branch *ptpx-to-merge*. You may want to checkout the *tom_mods* branch.

- The Linux driver is at `git://github.com/a-rubini/gnurabbit.git` as well as my own `git://gnudd.com/gnurabbit.git`.
- The source for this documentation is at `git://github.com/a-rubini/spec-demo.git`; a precompiled PDF is released in the *files* section of the OHWR project.
- The hardware and gateware for the switch is under SVN at `http://svn.ohwr.org/white-rabbit`.
- The hardware and gateware for the SPEC is under git at `git://ohwr.org/hdl-core-lib/wr-cores.git`.
- Documentation about the SPEC is under SVN at `http://svn.ohwr.org/spec`.

9 Acknowledgments

Most of the work related to the demo package has been performed by Tomasz Wlostowski with Maciej Lipinski, both in the control group in the beams department within CERN (Genève); and Grzegorz Daniluk of Elproma (Warszawa).

The *White Rabbit* project is a multi-company multi-lab collaboration hosted at `ohwr.org`. Many people have been involved in designing building and running the SPEC and, in particular, the White Rabbit Switch.