This document is OUTDATED!
How to setup a mini WhiteRabbit network with two SPEC cards (Master ↔ Slave)

Martin Brückner
Ralf Wischnewski

November 1, 2012
Version 2

Abstract

We explain how to obtain a running WhiteRabbit master-slave setup, made of two SPEC-cards. Some parts are based on this tutorial: `http://www.ohwr.org/projects/wr-cores/wiki/Wrpc_core`

1 Linux setup

After failing with some kernel versions, we finally installed Ubuntu 10.10 x86_64 with Linux kernel 2.6.35.
Furthermore the following packages should be installed :

- git
- subversion
- make
- texinfo (removes annoying errors, necessary for documentation only)
- minicom or screen (for the RS232 output)
- linux-headers-2.6.35-32-generic (choose the headers corresponding to your kernel)
- ia32-libs (not sure if really necessary).

gcc should be already installed (or will be installed with make).
You must install the Xilinx ISE design suite if you want to synthesize the FPGA-design.

2 Download the necessary WhiteRabbit files

Tools:

- hdlmake - A tool which collects all necessary hdl-files for a successful build

- lm32 toolchain - Compiler tool chain for the lm32 software

SPEC-card stuff: For this tutorial we first used the repository revisions which were available around March 21th 2012. With an exception on `wrpc-sw` (see the corresponding section) it works for more recent revisions, too.

- wr-cores - Contains the WhiteRabbit FPGA design
  Revision `33e6e9d5c01396ebcc799ff1efc72809ad44116b` date 03/20/2012
- spec-sw - Contains the host pc linux kernel module (for downloading the firmware)
  Revision `729ab34ca9b5f19428be51f9f20bf01a9fe8825d` date 03/12/2012
- wrpc-sw - Contains the software code for the lm32-cpu which is part of the FPGA design
  Revision `4266b6f96bd7f6b794030f156a3a1a42d8eb0190` date 03/19/2012

First of all you should create your working directory (eg. whiterabbit).

```
mkdir␣whiterabbit
cd␣whiterabbit
```

For the following subsections we assume that you are in this directory. If you want to checkout an old revision type `git␣checkout␣<checksum>`.

2.1 hdlmake

```
git␣clone␣git://ohwr.org/misc/hdl-make.git
export␣PATH=${PWD}/hdl-make:$PATH
```

You have to type the export-line for every new shell session or add it to your `~/.profile` or any similar file.

2.2 lm32 toolchain

```
wget␣http://www.ohwr.org/attachments/download/1133/lm32.tar.xz
tar␣xf␣lm32.tar.xz
export␣PATH=${PWD}/lm32/bin:$PATH
```

Like in 2.1, you have to type the export-line on every new shell session or add it to `~/.profile` or any similar file.

2.3 wr-cores

```
git␣clone␣git://ohwr.org/hdl-core-lib/wr-cores.git
cd␣wr-cores
git␣checkout␣wishbonized
```

2.4 spec-sw

```
git␣clone␣git://ohwr.org/fmc-projects/spec/spec-sw.git
```

2.5 wrpc-sw

There are different ways of downloading for older and newer wrpc-sw revisions.

This is for older revisions like `4266b6f96bd7f6b794030f156a3a1a42d8eb0190`. Unfortunately the branch `wrcore_v2` does not exist anymore.

```
git␣clone␣git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git
cd␣wrpc-sw
git␣checkout␣<old_revision_checksum>
git␣checkout␣wrcore_v2
git␣clone␣-b␣ptpx-to-merge␣git://gnudd.com/ptp-noposix.git
```

This wirks for the current version:

```
git␣clone␣git://ohwr.org/hdl-core-lib/wr-cores/wrpc-sw.git
cd␣wrpc-sw
git␣submodule␣init
git␣submodule␣update
```

3 Build everything

3.1 Build the FPGA design

First, hdlmake collects all necessary netlists and hdl-files from the repositories (Internet) and copies cores from the Xilinx installation directory.
On some Linux distributions problems with the different library versions can occur. While Xilinx is using its own libraries, the tools called by hdlmake (like svn, git, ...) need the system libraries. Additionally hdlmake requires a modified `XILINX` variable. The `/ISE/` part has to be removed and added afterwards. Therefore we modified the `XILINX` variable and cleared the `LD_LIBRARY_PATH` variable for the hdlmake call.

```
cd␣wr-cores/syn/spec_1_1/wr_core_demo/
bash
source␣/path/to/your/xilinx/installation/ISE_DS/settings64.sh
XILINX=${XILINX%/ISE}
LD_LIBRARY_PATH=␣hdlmake
XILINX=${XILINX}/ISE
make
exit
```

Finally, you get a bin file called `spec_top.bin`. This is the FPGA design file for both cards, master and slave.

3.2 Build the lm32 software

You need two different software files. One for the master and one for the slave. Start every build process with `make␣clean`.

```
cd␣wrpc-sw
make␣clean
make␣WRMODE=master
mv␣wrc.bin␣wrc_master.bin
make␣clean
make␣WRMODE=slave
mv␣wrc.bin␣wrc_slave.bin
```

Now there are two files: `wrc_master.bin` for the master spec card and `wrc_slave.bin` for the slave card.

3.3 Build the kernel module

```
cd␣spec-sw
make
```

If there are any errors regarding a `atomic.h` file, remove the corresponding include lines in the source code files.

4 Try it out

After that, load the kernel module.

```
cd␣spec-sw
insmod␣spec.ko
```

You will see that nothing happens. This is because the files have to be copied to the right place, so that the kernel module can find them. Since we just loaded the module, we can see exactly how the files should be named. See the `dmesg` output. And move/copy/link the files with the corresponding names to `/lib/firmware`.
Example:

```
cd␣/lib/firmware
ln␣-s␣/home/wischnew/whiterabbit/spec_top.bin␣spec-B0001.bin
ln␣-s␣/home/wischnew/whiterabbit/spec_top.bin␣spec-B0002.bin
ln␣-s␣/home/wischnew/whiterabbit/wrc_slave.bin␣spec-B0001-cpu.bin
ln␣-s␣/home/wischnew/whiterabbit/wrc_master.bin␣spec-B0002-cpu.bin
```

As you can see, we are using the same hardware design for both cards.

Now reload the kernel module

```
cd␣~/whiterabbit/spec-sw
rmmod␣-r␣spec.ko
insmod␣spec.ko
```

You should have now loaded the two SPEC cards. Try the `dmesg` output, `spec.ko` should report loading the correct files (hint: check the file lengths).

5 Add PPS signal output on master card

This section is outdated because in the new revisions the PPS-signal is activated on the mastercard by default.

For older revisions:
If you installed a DIO5Ch-FMC on your SPEC cards, you might be interested to see the PPS-signals from both master and slave. While the slave design has the PPS output enabled by default, we found the modifications given below to work. (Please, let us know if there's any potential problem with this solution).
To get a pps signal out of the master you have to modify `wrpc-sw/ptp-noposix/PTPWRd/protocol.c`. In function `doState()` there is a switch-case structure with the case `PTP_MASTER`. You can add there `pps_gen_enable_output(1);`. It should look like this afterwards:

```
case PTP_MASTER:
  pps_gen_enable_output(1);
  if(ptpPortDS->wrMode == WR_MASTER  && ptpPortDS->
      wrPortState != WRS_IDLE) { /* handling messages
      inside: handle()*/
```

Now recompile it as described in 3.2, copy the resulting file to /lib/firmware and reload the kernel module.

6 History

- Version 1 - Initial version
- Version 2 - Updated download information and chapter 5 marked as outdated