White Rabbit Switch: software repository

May 2011

Alessandro Rubini (rubini@gnudd.com)
Introduction

This package is meant to include all white-rabbit software. At this point, it includes part of it but it is being worked on.

Please note that the documentation itself is split in three different documents, this includes only part of the story.

1 The Software Package

The package hosts the minic and nic drivers for the WR switch, the user space code, build scripts and testing procedures. The build scripts and test procedures are documented in separate files (pdf and all the other formats are available).

To compile the documentation you are expected to have emacs, texinfo and pdftex installed. emacs is used to recreate the “prev”, “next”, “up” links used by texinfo.

While the documentation is mainly my work, most of the code here is not mine. Tomasz Włostowski and Maciej Lipinski wrote most of the vhdl and user-space code; Emilio Cota, with Tomasz, wrote the initial kernel space implementation; Miguel Baizan and Juan Luis Manas wrote the RTU code.

There are many more people involved in White Rabbit and I’m clearly misrepresenting most of them here. If you are involved and you are editing this file, please add your name in all the proper places.

1.1 Directory Layout

COPYING GNU GPL Version 2. This package is copyrighted by CERN and others, and is released as free software.

doc/wrs-software.in
  Source for this documentation.

doc/wrs-build.in
  Documentation for the build scripts.

doc/wrs-test.in
  Documentation for the testing code.

build/
  Build scripts used to recompile all the software, documented in doc/wrs-build*.

kernel/
  Device drivers and support files. Remember to set the LINUX environment variable and your CROSS_COMPILE prefix in order to compile successfully. If you compile with the build scripts this is done automatically for you.

userspace/
  Applications used in the switch itself. Again, you need CROSS_COMPILE to be set unless you follow the build scripts.

test/
  Test procedures, documented in doc/wrs-test*.

patches/
  The directory hosts patches for both the kernel and other packages (as needed by the build scripts). Some of the patches here have been spit out by "git format-patch".
1.2 Role of this document

While the overall software compilation is described in a separate document, developers usually need to compile just the sub-parts they are working with, or get some insight about what the software is doing.

This document, thus, collects notes about compilation of individual parts separately from the build scripts, as well as some information about the NIC driver and possibly (over time) other software components.

2 Compiling the Kernel

To compile the kernel, you need to retrieve the pristine sources for 2.6.35 and apply the patches for the switch. You can do that using git or with old-style tar+patch. See the section about how to get the sources with git or tar and patch.

Once you have the wrswitch kernel, you should cd to its main directory and compile in this way. Note that the CROSS_COMPILE prefix may be the one you built using buildroot or any other arm-linux-gcc. The CROSS_COMPILE variable must end with the hyphen that appears before "gcc" in the name of the actual compiler.

```bash
export ARCH=arm
export CROSS_COMPILE=/path/to/cross/arm-linux-
mke wrswitchv2_defconfig
make uImage
```

Finally, copy the uImage to your tftp server directory, as the boot-loader in the switch is pre-configured to get a file called "uImage" from your tftp server.

```bash
cp arch/arm/boot/uImage /tftpboot
```

2.1 Using Git

If you use git, you are expected to already have some copy of Linus’ official tree. Let’s say it’s in /usr/src/linux-vanilla.git. In that case, you should first check out 2.6.35 and then fetch the wrswitch branch. The git tree where I have the stuff is git://gnudd.com/linux-wrswitchv2/

What follows is one possible way, adding branches to the vanilla tree. (I use different names for the various items to help the non-expert git user in differentiating them, a cleaner approach would be using "wrswitchv2" for all items in the different contexts).

```bash
cd /usr/src/linux-vanilla.git
git remote add gnudd git://gnudd.com/linux-wrswitchv2/
git fetch gnudd
git checkout -b wr-kernel gnudd/wrswitchv2-dev
```

At this point, you checkout is a new branch that corresponds to the "wrswitchv2-dev" branch on my gnudd server.

Another option is creating a different checkout that uses data from the Linus tree you already have on your system. (Strictly speaking, you can avoid creating the "linus" remote to fetch it, but I feel this is clearer).

```bash
export GIT_ALTERNATE_OBJECT_DIRECTORIES=/usr/src/linux-vanilla.git/objects
mkdir /usr/src/linux-wr

cd /usr/src/linux-wr

git init

git remote add linus /usr/src/linux-vanilla.git/objects

git remote add gnudd git://gnudd.com/linux-wrswitchv2/
```
git fetch linus
git fetch gnudd
git checkout -b wr-kernel gnudd/wrswitch2-dev

Please note that if you don’t have a local "Linus" tree first, your fetch from gnudd will take several hours. You should first fetch from a mainstream server the official kernel to make this operation fast.

To make the "GIT_ALTERNATE_OBJECT_DIRECTORIES" setting persistent, you can save it to local repository:

```
echo /usr/src/linux-vanilla/objects > .git/objects/info/alternates
```

## 2.2 Using Tar and Patch

If you are not a git user, you might apply the patches, with this procedure, using your national mirror instead of ch.kernel.org. Let’s assume your starting current directory is the `wr-switch-sw` main directory

```
DOCDIR=$(/bin/pwd)
mkdir /usr/src/linux-wr
cd /usr/src/linux-wr
tar xjf linux-2.6.35.tar.bz2
mv linux-2.6.35/* linux-2.6.35/.??* .
rmrdir linux-2.6.35
for n in in $DOCDIR/patches/kernel/v2.6.35/*patch; do
cat $n | patch -p1
done
```

## 3 Compiling Kernel Modules

To compile the kernel drivers you need to run "make" in the `kernel/` subdirectory of the package, after setting up the proper environment variables:

- `LINUX` should point to the kernel directory. For example, export `LINUX=/usr/src/linux-wr`
- `CROSS_COMPILE` should be the same prefix you used to cross-compile the kernel. If `CROSS_COMPILE_ARM` is set, it is used as a default (the name `CROSS_COMPILE_ARM` is used in wrdev2 scripts)
- `ARCH` should be "arm", but the Makefile sets it if not predefined.

With the variables in place, after "make" you’ll have a few kernel modules, with extension ".ko", that should be copied to the target system and loaded. Let’s assume the target is `T=192.168.1.23`

```
scp $(find . -name '*.ko') root@$T:/wr/lib/modules
```

The previous command assumes you either know the root password of your private key is authorised in the target system. The latter happens if you build the filesystem using wrdev2 or the build scripts part of this package.
4 The NIC Driver

The kernel/nic/ directory includes the NIC driver for white rabbit. The has been laid out to be as understandable as possible, since I’m not the one who’s going to debug it. These notes are for the ones who will debug my crap. The published one will be different, hopefully.

4.1 The Makefile

You can "export WRN_DEBUG=y" to force all pr_debug and netdev_dbg into real printks (with KERN_DEBUG priority: you get them in dmesg or you need to raise the console priority to see them on the console).

4.2 The Naming Conventions

wrn_

  as a prefix is used for all functions and macros, so you immediately know if sth is ours or comes from elsewhere (as an exception, names of our own bits and registers are kept as I got them, the context should say sth about what they arr)

wrn

  is always a pointer to the wr-nic overall structure

ep

  is the pointer to the endpoint structure

dev

  is the pointer to the network device (device.c is an exception, is uses "netdev" instead, sorry for the confusion)

4.3 The NIC Files

The module is called wr-nic.ko which comes from several object files. This is the layout. Files which need to be fixed have FIXME inside and in this list as well.

..//wbgen-regs/

  The directory includes the register lists. See the README in there about how to recreate them.

nic-hardware.h

  This includes only #defines for magic project-wide constants. It then includes the wbgen-generated headers for individual register blocks.

wr-nic.h

  the header is used for sharing stuff between sources. Unlike the previous one, which it includes, it define kernel-level abstractions.

module.c

  registering and unregistering the module

device.c

  the actual probe and remove functions, and the driver structure

endpoint.c

  Talking with the actual endpoints, including the probe part. Mii/phy stuff is here, as well as the polling timer for link status.
nic-core.c
standard networking stuff: interrupts and so on

ethtool.c
ethtool operations

pps.c
the pulse-per-second mechanism. This might want to be registered in Linux using drivers/pps, but it must be looked at. This file is currently registering itself as a timesource.

timestamp.c
all the stuff related to timestamping

dmtime.c
calibration and readout
Table of Contents

Introduction ....................................................................................... 1

1 The Software Package ................................................................. 1
  1.1 Directory Layout ................................................................. 1
  1.2 Role of this document .......................................................... 2

2 Compiling the Kernel ................................................................. 2
  2.1 Using Git ........................................................................... 2
  2.2 Using Tar and Patch ............................................................ 3

3 Compiling Kernel Modules .......................................................... 3

4 The NIC Driver ............................................................................ 4
  4.1 The Makefile ....................................................................... 4
  4.2 The Naming Conventions ...................................................... 4
  4.3 The NIC Files ...................................................................... 4