# Introduction

In a tutorial on ni.com, I provided a library to use the rtl-based sdrs with both Labview for Windows and in a Labview RT environment. For those who don't know, rtl-sdrs are little small $20.00 USB dongles which are normally used for digital video reception on the PC. It is possible, however, to access the raw IQ data stream and therefore to use the dongle as a broadband RF receiver between 50MHz and 1.8GHz(model specific). While the bandwidth at 2.4MHz is tiny by today's professional standards, it's still the cheapest option to begin working with RF and SDRs. Please check out the original tutorials for more information and some background.

<https://forums.ni.com/t5/Example-Program-Drafts/Using-RTL-SDR-with-Labview-Chapter-1-Labview-on-Windows/ta-p/3538774>

<https://forums.ni.com/t5/Example-Program-Drafts/Using-RTL-SDR-with-Labview-Chapter-2-Labview-RT/ta-p/3538912>

I was recently asked if I could support an effort to use this driver on a LINX-based PI setup. At first, this seemed to be an easy project, but given the fact that I didn't understand the NI chroot environment (or chroot in general) in which the labview runtime runs, I severely underestimated the required effort. After some late nights, I was finally able to get a labview vi on the pi to get samples from an rtl-sdr connected on the pi's usb port. For posterities sake, I have documented my adventures through the bowels of linux :-)

I have divided the process up into separate parts. Part 1 deals with the configuration of the Pis host Linux. In this case, we will be using raspbian. Part 3 deals with configuring the National Instruments chroot environement. This gets a bit ugly at times, because some of the tools we need such as cmake and the boost libraries are not available as packages. Therefore we will be compiling these from source. Don't worry though, it sounds more difficult than it is. Part 5 shows a small little demonstration .vi running on the pi which tunes the rtl-sdr to a specific frequency, samples data, performs an fft and displays the results.

This tutorial assumes basic linux command line knowledge. It is possible to follow along if you've never worked on the command line, but it will be difficult for you to figure out what's gone wrong. Additionally, you should have a basic understanding of the linux file system and the editor vi.

I've delimited the command lines to executed by < at the start and > at the end. You can copy everything between those characters and paste it into the command line.

The source code for the demo project is available on github, along with the latest version of this document.

<https://github.com/albertlederer/LabviewLinxRTLSDR>

Last but not least, be prepared to invest some time. Some of the compiles take a long time. And should you find any mistakes in this tutorial, please let me know.

# Part 1: Getting the Pi Ready and Compiling the librtlsdr.so Library in the Host Linux

This tutorial assumes a clean image of raspbian. The version I used was "April 2017" with a release date of 2017-04-10 and kernel 4.4. Once the basic setup is completed (enable ssh, expand filesystem, set hostname, etc), run

<*sudo apt-get update && sudo apt-get upgrade*>

to upgrade all packages to the latest revision. In order to compile the rtl-sdr library, we will need to download some more packages so that all the required libraries are present and the compile will run through without a hitch. This is a bit different than on the myrio because we will be using the apt-get package manager.

<sudo apt-get install libusb-1.0-0-dev gcc git cmake binutils autoconf automake libtool libboost-all-dev pkg-config make>

This will take a while as there are quite a lot of packages to download. Go get some coffee or take a nap.

If you're not already in the home directory of the pi user, switch to that. Simply run <cd /home/pi> to get there.

Once this is done, we can clone the rtl-sdr repository using:

<*git clone git://git.osmocom.org/rtl-sdr.git*>

This will create a directory called /home/pi/rtl-sdr. That's where we will be compiling the rtl-sdr library for the first time so that the host linux has the kernel drivers available to access the rtl-sdr stick in the proper mode (sdr, not dvb-t).

<*cd rtl-sdr/*>

<*mkdir build*>

<*cd build*>

<*cmake ../ -DINSTALL\_UDEV\_RULES=ON -DDETACH\_KERNEL\_DRIVER=ON*>

<*make*>

<*sudo make install*>

<*sudo ldconfig*>

<*cd..*>

<*sudo cp ./rtl-sdr.rules /etc/udev/rules.d/*>

<*sudo reboot*>

Alright. So far so good. I suggested in the original post that you use "rtl\_test" to verify that the libary works and that applications using the library can see the dongle. If all is well, continue on. If not, check to make sure that the dongle is properly connected to the pi, that you have performed the final reboot, and that the compile was performed successfully.

# Part 2: Installing the NI Software

Start Labview 2014 and install the NI software on the PI using Tools-->Makerhub-->Linx-->Linx Target Configuration. The process is straight forward, so I won't cover it here in detail. There is enough support documentation at the Labview Makerhub to get this going.

# Part 3: Getting the NI chroot environment ready

Go to /home/pi and create a directory called natinst. Inside this directory we will compile all the tools we will need later.

Change to /home/pi/natinst using the followng command:

<*cd /home/pi/natinst/*>

Now we will download the cmake and the boost source code. These tools are necessary to compile the rtl-sdr software, but they are not available as packages from the NI repository.

Type in the following commands:

<*wget* [*https://cmake.org/files/v3.8/cmake-3.8.1.tar.gz*](https://cmake.org/files/v3.8/cmake-3.8.1.tar.gz)>

<*wget* [*https://dl.bintray.com/boostorg/release/1.64.0/source/boost\_1\_64\_0.tar.gz*](https://dl.bintray.com/boostorg/release/1.64.0/source/boost_1_64_0.tar.gz)>

<*git clone git://git.osmocom.org/rtl-sdr.git*>

IMPORTANT: Now change to a directory NOT part of the chroot environment. If you don't do this, you will get some funky "parent directory" errors and all your compile attempts will fail. I used /srv/chroot/labview. I found this out by accident and it took me a while to figure out why subsequent builds weren't working. I think this is a rights issue with the host<->chroot file system. If anyone has an explanation or a fix for this, please let me know.

<cd /srv/chroot/labview>

Now we have all the tools we need. At this point it's time to switch into the NI chroot environment using:

<*sudo schroot --run-session -c lv*>

This will drop you into a shell inside the chroot environment.

Now we will install all the packages available.

<*opkg update*>

<*opkg upgrade*>

<*opkg install libusb-1.0-dev gcc gcc-symlinks git binutils autoconf automake libtool pkgconfig make packagegroup-core-buildessential-dev packagegroup-core-buildessential*>

Again, this takes forever.

Ok, now drop into the /home/pi/natinst directory.

Run

<*tar -xvf cmake-3.8.1.tar.gz*>

<*tar -xvf boost\_1\_64\_0.tar.gz*>

In the cmake folder run:

<*./bootstrap*>

<*make*>

<*make install*>

This is again a lengthy process. Go drink some coffee.

Drop back to /home/pi/natinst and go into the boost directory.

In the boost folder run:

<*./bootstrap.sh*>

<*./b2*>

<*./b2 install*>

This is another lengthy process. Drink some more coffee (or take a nap. This really takes a long time.)

# Part 4: Compiling the rtl-sdr library inside the NI chroot environment

Now, we should finally be ready to compile the rtl-sdr library.

Go back to /home/pi/natinst and go into rtl-sdr. <*cd /home/pi/natinst*>

<*mkdir build*>

<*cd build*>

<*cmake ../ -DINSTALL\_UDEV\_RULES=ON -DDETACH\_KERNEL\_DRIVER=ON*>

<*make*>

<*make install*>

Once that's done, run

<*vi /etc/ld.so.conf*>

Add the following line to the end of the file:

/usr/local/lib

Exit vi using :wq

Now run <*/sbin/ldconfig*> to rebuild the shared library cache.

If everything worked out ok, you should now (finally) be able to run rtl\_test INSIDE the chroot environment. Give it a try. If that works, we're done, and you will be able to use the librtlsdr.so.lib as expected.

# Part 5: Labview Demo Project

I have included a small labview project to get you started working with the library. The project structure is very limited and is meant to provide a starting point for your own adventures. I’m going to briefly explain the project and what you have to do to get it to work.

## The Project

|  |  |
| --- | --- |
|  | This is the basic project structure.   * The first thing you should notice is the Raspberry PI 2 B target. If you right click on this and select “Properties”, you can change the IP address to match your PI and change the name of the target (note that this is the target name only, not the PI’s hostname). * There is also a vi called Pi\_Main.vi. It holds the demo code. The auto-populating folder called RTLSDR contains the subvis necessary to access the sdr. |

## RTLSDR

|  |  |
| --- | --- |
|  | I’ve expanded the RTLSDR folder so you can see all the vi’s you can use to work with the RTL SDR.   * Get Vis are used to get current settings of the device and how many devices are connected to the system. * Read VIs are used to read samples from the device. Read Async does not work as of yet. I’m working on it. * Set Vis are used to set properties, such as the tuned frequency and sample rate (bandwidth. * The controls folder contains support controls and type definitions. You should leave those alone. |

## PI\_Main

|  |
| --- |
| C:\Users\TheDoctor\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Snippet.png |
| This is the Pi\_Main.vi block diagram. As you can see, it’s very simple. It’s meant as a starting point for your projects. There is a lot of yellow, which is as it should be. The comments should be sufficient to understand the basic principle of the library. |

|  |
| --- |
|  |
| This is the Pi\_Main.vi. There’s not really much here yet, but that’s for you to fill out. You can see the indicator show the number of rtl-sdr devices connected to the system. The bigger part of the front panel is taken up by the spectrum display. The frequency scaling is correct, however the amplitude scaling is meaningless. |

# Part 6: Closing Thoughts

I hope that someone, somewhere will find this walkthrough useful. Should you notice any bugs, errors or some such, please let me know. I’m always open for constructive criticism.

If you’re looking for some project ideas, I’ve got a few to get you started:

* Build an FM radio.
* Build a scanner that scans a frequency range for new signals.
* Look for signals on the 433 ISM band.
* Build a decoder for NOAA weather satellite data.

These aren’t glorious ideas that will change to course of history, but it’s usually a good idea to start with baby steps so you get a feel for the system and what it can do.