Low Cost SDR Spectrum Analyzer and Analog Radio Receiver Using GNU Radio, Raspberry Pi2 and SDR-RTL Dongle

E. G. Sierra*, G. A. Ramírez Arroyave*†
*Universidad Católica de Colombia,†Universidad Nacional de Colombia
{egsierra, garamirez}@ucatolica.edu.co

***Abstract*—This paper presents the implementation and test of a low cost Software Defined Radio based on the Raspberry Pi2 computer, the RTL-SDR dongle radio receiver, and the GNU Radio software. Key aspects regarding installation and configuration of GNU radio and RTL-SDR drivers on Raspbian Operating System are addressed. The resulting device can demodulate AM and FM signals in the frequency range from 30MHz to 1.7GHz and also can meet the function of Spectrum Analyzer in moderate sensitivity demanding applications.**

***Index Terms*—Software Defined Radio, GNU Radio, Raspberry Pi 2 board computer, RTL-SDR dongle, Spectrum Analyzer, Analog demodulation.**

I. Introduction

Software defined radio - SDR is a communications system paradigm in which many of the traditional functions of the radio transceiver, more commonly signal processing, are carried out by software commands rather than by hardware deployment either analog or digital. One of the seminal papers on the topic is the celebrated work of 1995 by Mitola [1] in the context of GSM/CDMA standards for wireless mobile communications in which alternative architectures are defined for the development of SDR and the further step of Cognitive Radio - CR which even today is matter of technical discussion, especially for the difficulties that arise trying to achieve reconfigurable radio front-ends (Antennas and filters). Other aspects of SDR and CR like regulatory issues, service provision, and convergence, usually overlooked in technical literature are explored in [2].

From those early days when dedicated processors and specific mission software where the only practical alternative to the realization of SDR, computational power has increased and costs have shrunken according to the expectations [3], this factors combined have allowed recent developments of SDR and CR on general purpose hardware and software platforms outside the world of military and mobile wireless industry and hence raised interest on the topic to a broad audience. An historical perspective with recent developments and current challenges of SDR can be found in [4] where the topicality of the issue is exposed.

Academy is not oblivious to this trend, traditionally, most of the topics in communication systems and signal processing courses which require a vast set of abstract mathematical tools and models related to analog/digital modulation/demodulation are better understood by means of simulation, this is usually carried out with Simulink, Python/Scilab/Matlab/Octave code, or similar tools. Although, recently SDR construction tools like GNU radio are becoming a key enabler for teaching as reported in [5, 6], given the relatively easy and cheap transition from simulation to real time processing of real world signals.

Even though this contribution derives from a class project at the undergraduate level, at the far end it aims to show the potential of SDR and the proposed combination of low cost tools not only in the engineering curriculum but also as a serious tool-set for the development of projects at various educative levels and even in industry start-ups. Focus herein is on the development of a very low cost SDR spectrum analyzer with FM/AM demodulation capabilities using the Raspberry Pi2 computer [7], the RTL-SDR radio receiver [8], and the GNU Radio software [9].

Similar efforts are vastly reported on literature, specifically the SDR reception of FM signals which of course can be achieved in several ways, for example using FPGAs as in [10] where a complete analysis of feed-forward and feed-back alternatives for the discrete time FM signals demodulation is carried out, or as in [11] where RTL-SDR dongle is used as radio front end and a PC is used for the signal processing stage using simple and straightforward signal manipulations implemented directly in Python language, or for example as in [12] where an FM receiver is implemented using the professional grade receiver Universal Software Radio Peripheral - USRP by Ettus research [13] which is used to receive the radio-frequency - RF signal, down-convert and discretize it and then send a digital low-pass signal to a PC where GNU radio does the demodulation part.

A related work is presented by Danymol et al in [14], where RTL-SDR, Raspberry Pi model A and GNU radio are used in a very different way to our proposition, inasmuch as they use the Raspberry Pi only to retransmit the RTL-SDR raw data to the PC where GNU radio companion is running and executes the demodulation. Another recent work reported in a blog by Back [15] does have many common steps to our work but greatly differs in the intended application as there they concentrate on the recover of Mode S transponder signals from air navigation

978-1-4673-8451-3/15/$31.00 ©2015 IEEE

systems using the gr-air-modes software [16] instead of GNU radio companion.

Also, spectrum analyzers based on similar tools (RPi2 and RTL-SDR dongle) are reported on the Internet, for example [17] where the author has developed his own Python code based on Numpy [18] to plot the FFT or waterfall spectrum of the captured signal, even though this is a very nice approach it lacks the scalability and extended capabilities of GNU radio. Similarly there are some reports on the Internet, most of them really shallow, that show the use of the gqrx software [19] which is based on GNU radio, or the SDR-Sharp software [20] which is a widespread SDR receiver option running on windows platforms, in conjunction with RPi board as a signal relay to a personal computer where processing is done.

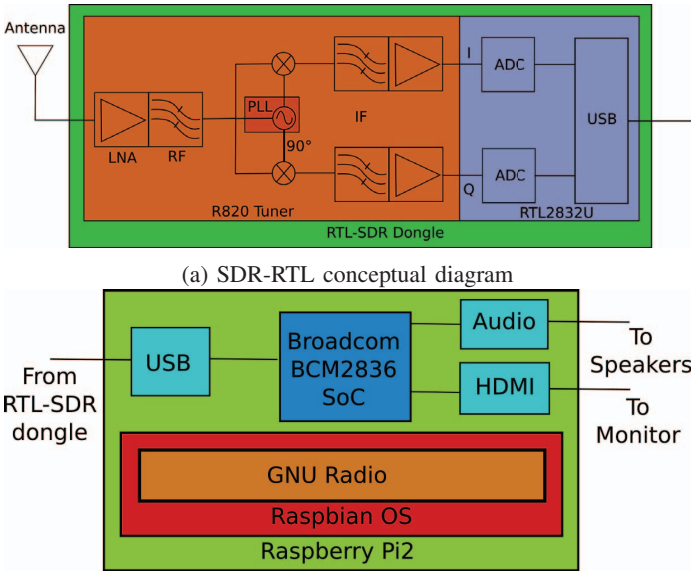
In contrast to the aforementioned approaches this project focuses on doing all the processing on the Raspberry Pi2 computer running a standalone version of GNU radio companion, using the RTL-SDR as signal source, this setting greatly reduces cost and with all the possibilities enabled by GNU radio allows with little effort many different applications by means of simple block manipulations on its graphical interface, or in more elaborated cases the development and use of personalized processing blocks, hence this contribution is a step towards autonomous SDR and CR systems that do not rely on PCs for the real time signal processing and/or for the execution of the cognitive algorithms involved in CR.

The remaining sections of this paper will discuss the design and setup of the components used in the project, the results of the experiments performed, and finally a section with the conclusions and aspects of future work.

II. Design and previous setup

The realization of the SDR herein is based on a simple retractable monopole antenna connected to the RTL-SDR dongle that serves as receiver of the RF signals in the band from 0.03 to $1.7GHz$ and delivers digital I and Q channels via USB connection, the RTL-SDR is directly connected to one of the USB ports of the raspberry Pi2 running Raspbian OS where a session of GNU radio demodulates the incoming signal and sends the audio raw base-band data to speakers and the GNU radio companion screen to a monitor via HDMI, the conceptual diagram of connections is presented in fig 1 where a high level functional view of RTL-SDR and Raspberrry Pi2 are drawn.

The Raspberry Pi2 computer, launched in February 2015, was chosen for this application due to its computational power (it can run the full range of ARM GNU/Linux distributions and even is planned to run Microsoft Windows 10), and its relative low cost (listed at 35USD in US and EU retailers, price can vary depending on the retailer and country). The Raspberry Pi2 is based on the Broadcom BCM2836 System on a Chip - SoC which contains a 900MHz quad quad-core ARM Cortex-A7 CPU and a VideoCore 4 dual-core GPU, is also packed with 1 GB of RAM, Ethernet, HDMI, USB and general purpose input and output - GPIO ports, among other conveniences.



(a) SDR-RTL conceptual diagram

(b) Rpi2 conceptual diagram

Fig. 1: Conceptual diagram of the implemented SDR

The RTL-SDR dongle is a very popular device among the Ham radio community, originally thought as receiver for the Terrestrial version of the Digital Video Broadcast Standard - DVB-T, the dongle has become a useful low cost RF front end for hobbyists and practitioners around the world since it was hacked, unfortunately no official datasheet is available and related information has been collected by reverse engineering, most of it is summarized in [21, 8]. The RTL-SDR dongle has as main characteristics a radio front based on the RafaelMicro R820T tuner usable in the range from 24MHz to 1.76 GHz, with an input RF port of 75Ω, noise figure of 3.5dB, a worst case sensitivity of -81.5dBm, selectable intermediate frequency - IF around 4MHz; the core of the RTL-SDR is the Realtek RTL2832U integrated circuit which is a DVB-T Coded-OFDM demodulator, this chip is in charge of sampling and delivering digitalized bit streams via USB port, each sample is carried out with an 8 bits ADC at a maximum sampling rate of 3.2MS/s, these factors greatly limit the operation of the dongle in demanding applications. The RTL-SDR was chosen for this project because of its low cost, no need for internal setup in contrast to professional grade devices as USRP or similar, and given that our application only functions in receiving mode.

Maybe the biggest limitation in this approach, and a recurring one in literature, is the lack of a reconfigurable or ultra-wideband antenna with good performance across the whole examined band. In this work we used a humble retractile monopole that provides with a moderately good match to the 75Ω port only in the VHF band, characterization was not done to this antenna.

Finally we opted for GNU radio over Raspbian for the Software part of this SDR, the main reason for this choice, as outlined in the introduction, is the easiness and the intuitive

character of connecting blocks instead of writing code in a first approach to SDR, and also the vast opportunities for scaling up in future developments.

The steps for the configuration of each component to run properly on the Raspberry Pi2, and the main issues that can arise are depicted in the subsections that follow. This is important because in many blogs and sites around the web many people has their own version on how to achieve this objective but most are inaccurate/incomplete or either assume this previous setup is already done.

A. Setting up the Raspberry Pi2

The first thing to do with the Raspberry Pi2 out of the box is to set up an Operating System, several OS options exist as for example those based on the Linux Kernel such as Raspbian, Pidora, Arch Linux, OSMC, and OpenELEC, there are other options as RISC OS which is an operating system designed from scratch specifically for the ARM processor, and even a version of Windows 10. Raspbian, the recommended beginner's choice from the RPi team, was chosen due to its support and documentation, moreover given the availability of the Debian jessie packages which are supported by Raspbian makes it feasible to run GNU radio with few complications.

A typical and easy way to install an OS on the RPi2 is by means of New Out Of the Box Software - NOOBS package which comes in full and lite versions, another option is to download the image from the Raspbian page and install it manually. If you chose the manual option as done in this work take into account that to install Raspbian it is necessary to have a micro SD card of minimum 8GB and Internet connexion available to the RPi2, also it is important to keep in mind that when installing Raspbian for the first time the micro SD must be expanded, this setting is done in the configuration tools of Raspbian.

```
raspi-config
# Setup Options
Expand Filesystem       # Ensures that all of
the SD card storage is available to the OS
```

Listing 1: First time configuration of Raspbian

After installing Raspbian and rebooting the Raspberry the next step is to install GNU radio, and the drivers for the SDR-RTL dongle.

B. Installing GNU radio and its dependencies on the Raspberry Pi2

The steps described herein are based on [15], all commands must be executed as superuser, in the first place the packets of jessie must be downloaded, this is done by editing the $sources.list$ that is a text archive found in the directory $/etc/apt/$, to do so simply add the line shown in listing 2 in the mentioned archive and save changes. Take care and do not modify the rest of the file.

```
deb http://archive.raspbian.org/raspbian jessie
main
```

Listing 2: Installation of jessie packages

Thereafter the Raspbian repositories must be updated and then it is possible to install GNU Radio run-time and development environment, this is achieved executing the commands listed in 3.

```
apt-get update
# And the instruction to install GNU radio
apt-get install gnuradio gnuradio-dev
```

Listing 3: Instalation of GNU radio

At this point there were some problems previous to the installation of GNU radio due to the locale variables given that the Spanish language character set is not default, if you have similar issues the best workaround is to install them executing the command shown in listing 4.

```
apt-get install locales
```

Listing 4: Installation of locales

Once GNU radio is correctly installed, the next step is to set up the RTL-SDR drivers in order to properly use it as a signal source in the form of a block of GNU radio companion. To do so the SDR-RTL dongle must be prevented from being recognized at boot time by the OS as a DVB-T tuner, in this vein the file $raspi-blacklist.conf$ located in the directory $/etc/modprobe.d/$ must be created/modified adding the lines of listing 5.

```
blacklist rtl2832
blacklist r820t
blacklist rtl2830
blacklist dvb_usb_rtl28xxu
```

Listing 5: Black list SDR-RTL

Finally the RTL [21] components, that allow the use of RTL-SDR as a source block within GNU radio are installed using the commands listed in 6.

```
apt-get install rtl-sdr gr-osmosdr
```

Listing 6: Installation of GNU radio RTL-SDR drivers

C. Testing RTL-SDR and GNU radio on the Raspberry Pi2

In figure 2 the final assembly except the monitor is shown. After the set up stage described above, the first test to prove everything behaves as expected was to use the built-in functionalities of the RTL-SDR library which can be executed from the Raspbian command line and comprises a wide-band FM radio receiver, and a command to test the dongle's operating frequencies and sample rate.

The first is the RTL_FM program which can initialize the RTL dongle, tune a specific frequency, and send the decoded audio to a file or to an available player, also callable from the command line, as the alsa aplay or the sox play. The second is the RTL_SDR program that makes available the raw I-Q data for extensive analysis. For example with the option '-t' gives the possible tuning range, whereas the option '-s' allows to define the sampling rate and verify if there are dropped samples. These commands are used as presented in code fragment 7.

Fig. 2: Hardware Assembly

```
rtl_fm -f 98.5e6 -M wbfm -s 200000 -r 48000 - |
aplay -r 48k -f S16_LE
rtl_test -t
rtl_test -s 3.2e6
```

Listing 7: Test of RTL-SDR on RPi2

When this straightforward test is executed the RPi2 becomes a simple specific mission SDR dedicated to receive, decode, and play FM signals, as its processing capacities are more than enough for this application audio quality is really good and hence mono output can be clearly listened. Nevertheless, with the rtl_test command option '-t' there was a compatibility problem related to the R820T tuner inside the RTL-SDR and frequency sweep was not available to characterize the tune range of the receiver, while the '-s' option allowed to observe that the maximum sample rate that does not drop samples is 2.8MS/s.

III. Results

After setting up all the hardware and software components, the next step consists of running GNU radio companion, the graphical interface of GNU radio, on the RPi2 board with the aim of directly build, edit, and execute SDRs by a simple block diagram manipulation.

Two main tests were performed, in the first place the listening and "visualization" of commercial FM radio stations was performed and in second case a comparison to a test laboratory spectrum analyzer was carried out in different frequency bands.

A. FM listening

A simple radio receiver was built in GNU radio companion as shown in figure 3. There can be seen how the RTL-SDR is used as signal source, and FFT based frequency and waterfall sinks are used to fulfill the spectrum analyzer functions, a standard Wide-Band FM receiver of GNU radio is employed to detect FM modulated signals, miscellaneous blocks are employed to filter and resample signals.

Some remarks regarding this diagram implementation are worth, the use of QT GUI sink blocks is advised given their superior performance compared to WX GUI based ones, even if the second ones are widely used in examples around the literature these are strongly discouraged as they present serious lags when used in the present hardware configuration, moreover they will be discontinued in the future releases of GNU radio.

The block WBFM receive in charge of executing the FM quadrature signal demodulation can be replaced by other options as a properly developed C++/Python detector, for example using the feed-forward and feedback architectures exposed in [10] but this is subject of a more deep treatment of GNU radio which is beyond the scope of this first approach.

With the setup of fig 2 and the block diagram of fig 3 running on it, the allocated bandwidth to the commercial FM radio was swept, and the available stations were easily tuned and demodulated with acceptable sound quality as subjectively compared with a commercial radio receiver.

Figure 4 shows the received power spectrum along with the spectrogram in the frequency range from 90 to 91.5MHz where there can be appreciated the presence of two stations at 90.4 and 90.9 MHz, both about 20dB above the noise floor of the receiver.

B. Comparison of the RTL based SDR sensitivity with a Spectrum Analyzer

In the second test performed the proposed assembly is compared to the available Gw Instek spectrum analyzer 'GSP 810' which spans a frequency range from 150kHz to 1GHz, with resolution bandwidths - RBW of: 3K, 30k, 220k, and 4MHz, this instrument also has the capability of demodulate WBFM signals of 120kHz frequency deviation.

The first difference that appears obvious is the frequency span that favors the low cost assembly by a 1.7:1 factor on the upper band, moreover on the lower portion of the spectrum, there is a clear disadvantage for the RTL-SDR which can not go below 30MHz whereas a lab test equipment should be well under this value. This of course is a limitation of the radio front used, based on the RTL-SDR that is designed to work in the VHF-UHF portion of the electromagnetic spectrum and not as a high sensitivity ultra-wideband instrumentation receiver, so for low frequency applications another tuner should be used.

A direct comparison was performed in the allocated band to commercial FM stereo radio, results are shown in figure 5, here can be watched that the curves behave in a very similar way, despite the sensitivity difference.

A second test was performed in other frequency bands to compare the frequency span of receivers, among others, carriers were chosen in 800 MHz where SDR equipment can detect a carrier only 10dBm above its noise level, and also in the digital terrestrial television - TDT band of 480 MHz where the carrier is detected but due to the previously mentioned limitations only a 2MHz portion of spectrum can be seen. Results around 880 MHz GSM band are shown in figure 6 showing agreement in both instruments.

Finally with the aim of comparing the noise sensitivity of both receivers a central frequency of 150MHz was chosen, due to the fact that few interference was present in near bands. There was seen that the analyzer can resolve signal levels of

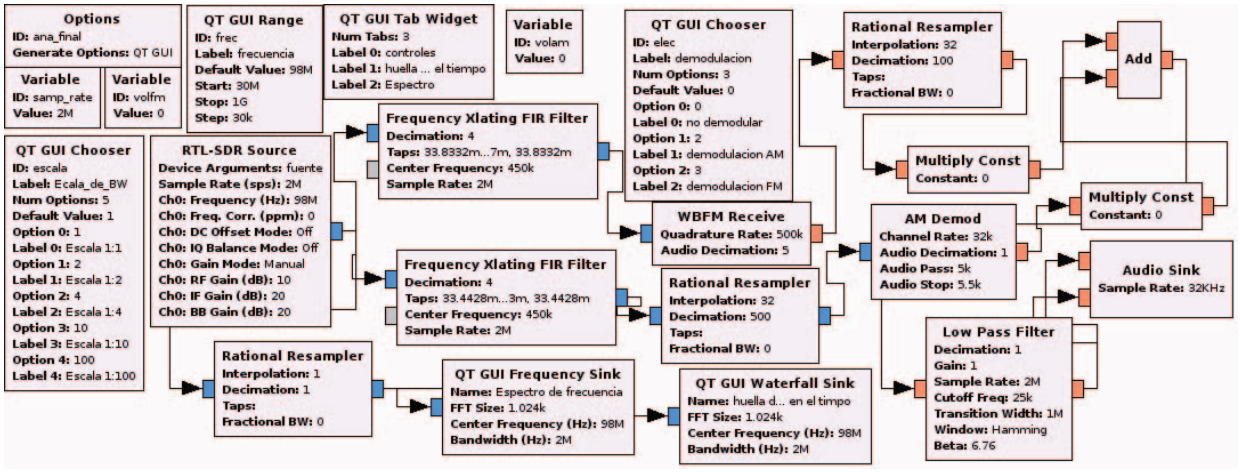


Fig. 3: GNU radio companion block diagram of simple Spectrum Analyzer and FM detector

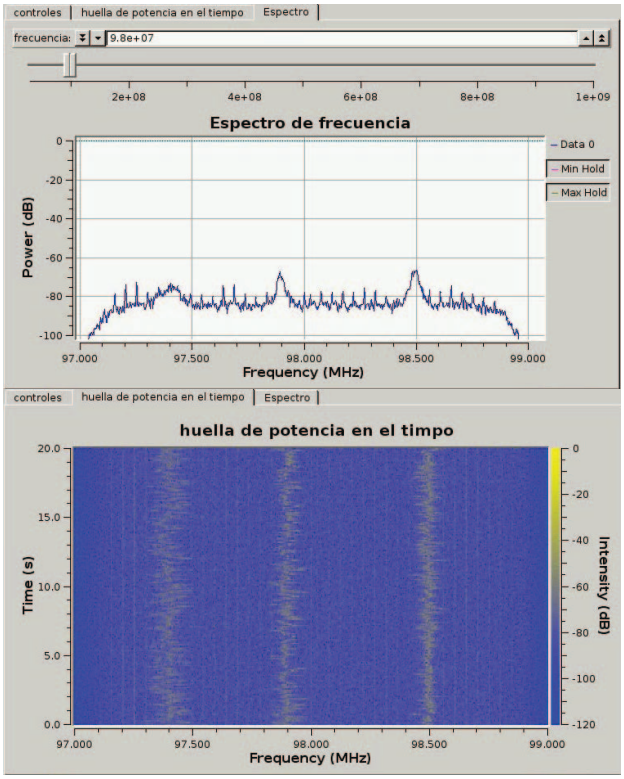


Fig. 4: Portion of the received Power spectrum in the FM radio band

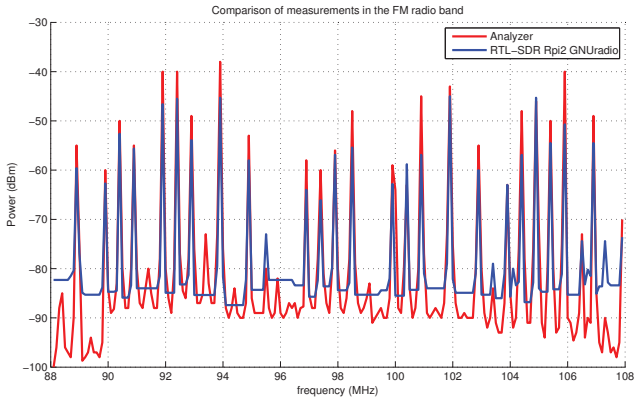


Fig. 5: Analyzer vs Raspberry

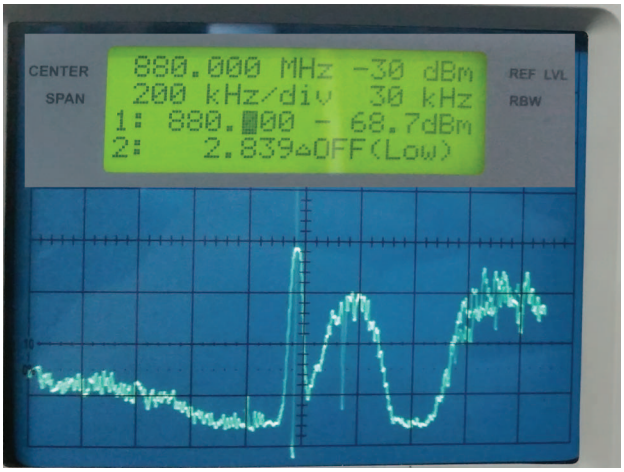
approximately -100dBm whereas the RTL module has a noise immunity about -83dBm.

IV. Conclusions

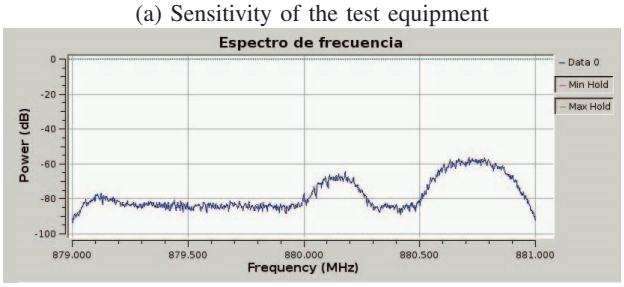
It is evident that installing GNU radio and RTL-SDR drivers on Raspberry Pi have many steps and hence is prone to errors which affect the performance of the hardware and developed applications, an easy and strongly encouraged way to avoid this errors should be the creation of a Linux script for automatically installing the applications and its dependencies on RPi2.

The use of GNU radio has proven very useful in the learning and understanding of digital processing of communication signals, with the additional advantage of being of GPL license and hence maintaining costs low related to other considered options. Likewise, the use of the Raspberry Pi2 computer has proven being powerful enough to run GNU radio companion while also executing the real time demodulation of FM signals and allowing the visualization of FFT power spectral density and waterfall frequency-time representation. Also, although sensitivity is poor compared to laboratory/professional grade measurement devices, the results achieved with the SDR-RL receiver are more than satisfying and prove a great value for its cost.

This project has shown the realization of a completely standalone SDR based on public domain and low cost tools. Many other applications should use similar setups of hardware and software, with further improvements more complex systems can be realized. Hardware extensions as the use of USRP are suggested to achieve transceivers that allow the simultaneous



(a) Sensitivity of the test equipment



(b) Sensitivity of the RTL-SDR RPi2

Fig. 6: Sensitivity comparison

transmission and reception of modulated signals. Also, more complex signals can be tested by means of the capabilities of GNU radio software not explored in this approach.

Reconfigurability is an obvious next step that must be pursued, and the use of this flexible hardware and software platforms can allow much more complex systems with improved performance at a very appealing cost benefit.

References

[1] J. Mitola, "The software radio architecture," *IEEE Communications Magazine*, vol. 33, no. 5, pp. 26–38, 1995.

[2] W. H. W. Tuttlebee, "Software-defined radio: Facets of a developing technology," *IEEE Personal Communications*, vol. 6, no. 2, pp. 38–44, 1999.

[3] C. Mack, "Fifty years of moore's law," *Semiconductor Manufacturing, IEEE Transactions on*, vol. 24, no. 2, pp. 202–207, May 2011.

[4] R. G. Machado and a. M. Wyglinski, "Software-Defined Radio: Bridging the Analog-Digital Divide," *Proceedings of the IEEE*, vol. 103, no. 3, pp. 409–423, 2015.

[5] L. S. Nagurney, "Software defined radio in the electrical and computer engineering curriculum," *2009 39th IEEE Frontiers in Education Conference*, pp. 1–6, 2009.

[6] S. Katz and J. Flynn, "Using software defined radio (SDR) to demonstrate concepts in communications and signal processing courses," *Proceedings - Frontiers in Education Conference, FIE*, pp. 1–6, 2009.

[7] J. L. E. Upton, R. Mullins and A. Mycroft. (2015) What is a Raspberry Pi. [Online]. Available: "https://www.raspberrypi.org/help/faqs/#introWhatIs"

[8] H. K. K. C. V. S. Markgraf, D. Stolnikov and H. Welte. (2015) What is RTL-SDR. [Online]. Available: "http://www.rtl-sdr.com/about-rtl-sdr/"

[9] M. B. M. E. T. R. E. Blossom, J. Corgan. (2015) GNU Radio Overview. [Online]. Available: "http://gnuradio.org/redmine/projects/gnuradio"

[10] M. Rice, M. Padilla, and B. Nelson, "On FM demodulators in software defined radios using FPGAs," *Proceedings - IEEE Military Communications Conference MILCOM*, no. 4, 2009.

[11] B. Uengtrakul and D. Bunnjaweht, "A Cost Efficient Software Defined Radio Receiver for Demonstrating Concepts in Communication and Signal Processing using Python and RTL-SDR," *Digital Information and Communication Technology and it's Applications (DICTAP), 2014 Fourth International Conference*, pp. 394–399, 2014.

[12] a. F. B. Selva, a. L. G. Reis, K. G. Lenzi, L. G. P. Meloni, and S. E. Barbin, "Introduction to the software-defined radio approach," *IEEE Latin America Transactions*, vol. 10, no. 1, pp. 1156–1161, 2012.

[13] M. Ettus. (2015) About Ettus Research. [Online]. Available: "http://www.ettus.com/about"

[14] G. R. Danymol. R, Ajitha. T, "Real-Time Communication System Design using," in *2013 International Conference on Advanced Computing and Communication Systems (ICACCS -2013)*, 2013.

[15] A. Back. (2015) Taking the Raspberry Pi 2 for a Test Drive with GNU Radio - Installing GNU Radio and receiving aircraft radar with a USB TV tuner. [Online]. Available: "http://www.rs-online.com/designspark/electronics/eng/blog/taking-the-raspberry-pi-2-for-a-test-drive-with-gnu-radio-2"

[16] N. Foster. (2015) gr-air-modes Gnuradio Mode-S/ADS-B radio. [Online]. Available: "https://github.com/bistromath/gr-air-modes"

[17] T. Dicola. (2015) freqshow Rpi RTL-SDR frequency scanner. [Online]. Available: "https://learn.adafruit.com/freq-show-raspberry-pi-rtl-sdr-scanner/overview"

[18] Numpy.org. (2015) What is Numpy. [Online]. Available: "http://www.numpy.org/"

[19] R. N. A. Csete. (2015) gqrx sdr receiver powered by the gnu radio sdr framework and the qt graphical toolkit. [Online]. Available: "http://gqrx.dk/"

[20] C. Gommel. (2015) Airspy sdr sharp. [Online]. Available: "http://airspy.com/download/"

[21] D. S. S. Markgraf and Hoernchen. (2015) Osmocom sdr. [Online]. Available: "http://sdr.osmocom.org/trac/wiki/rtl-sdr"