TÍTULO: A Digital transmission and reception techniques with Software Defined Radio

AUTOR: Jesús Adrián Crespo Ortiz

TITULACIÓN: Grado en Telemática

TUTOR: dr inż. Łukasz Januskiewicz

UNIVERSIDAD: Lodz University of Technology

CENTRO: Faculty of Electrical, Electronic, Computer and Control Engineering

PAÍS: Polonia

Fecha de lectura: 28 de septiembre de 2016

Calificación:

El Coordinador de Movilidad,
Bachelor of Engineering Thesis

A Digital transmission and reception techniques with Software Defined Radios

Adrian Crespo

Student’s number: 901521

Supervisor:
dr inż. Łukasz Januszkiewicz

Łódź, 2016
Adrian Crespo
*Digital transmission and reception techniques with Software Defined Radio*
ABSTRACT

The purpose of this project is to learn the features of GNU Radio software package and to understand transmission and reception techniques through it. The aspects explored are the analog and digital transmissions. The results of this analysis can be used to understand how modulations, codifications, sample rates and filtering work in a real environment by using Software Defined Radio circuits and verifying their properties. This projects requires some basics of programing in Python and knowledge of signals and systems as well as digital signal processing. The exercises also require to know some concepts of signal classification, systems and filters classification, knowledge of some mathematical concepts such as the Z-transform and the relationship between the Z-transform and Discrete Time Fourier Transform, and at last, the student should know some basics of sampling: band-limited signal, the sampling theorem, conversion between analog and digital signals and vice versa.
RESUMEN

El objetivo de este proyecto es aprender las características del paquete de software GNU Radio y comprender las técnicas de transmisión y recepción a través de él. Los conceptos explorados son las transmisiones analógicas y digitales. Los resultados de este análisis se pueden utilizar para entender cómo los distintos conceptos sobre modulaciones, codificaciones, frecuencias de muestreo y filtrado funcionan en un entorno real mediante circuitos desarrollados con técnicas de radio definida por software (conocido en inglés como “Software Defined Radio”), así como la verificación de sus propiedades. Este proyecto requiere algunos conceptos básicos de programación en Python y conocimiento de señales y sistemas, así como de procesamiento de señales digitales. Los ejercicios también requieren conocer algunos conceptos de clasificación de señales, sistemas y filtros, como también el conocimiento de algunos conceptos matemáticos tales como la transformada Z y la relación entre la transformada Z y la Transformada de Fourier de Tiempo Discreto. Finalmente, el estudiante debe saber algunos conceptos básicos de muestreo: señal de banda limitada, el teorema de muestreo, conversión entre señales analógicas y digitales y viceversa.
Adrian Crespo

*Digital transmission and reception techniques with Software Defined Radio*
ACKNOWLEDGEMENTS

Thank you to dr. hab. inż. Sławomir Hausman and dr. inż. Łukasz Januszkiewicz of the Instytut Elektroniki, Politechnika Łódzka, for their guidance and support throughout this study, and especially for his confidence in me.

Thanks you to Prof. dr. Rafael Herradón Diez, Deputy Director of External Relations and Students of the School of Telecommunications Systems and Engineering of the Technical University of Madrid (ETSIST-UPM).
Adrian Crespo

*Digital transmission and reception techniques with Software Defined Radio*

To my parents and my sister.

*To Leandro de Haro Ariet, in memoriam*
### TABLE OF FIGURES

<table>
<thead>
<tr>
<th>Image</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Desktop displaying the default wallpaper</td>
<td>18</td>
</tr>
<tr>
<td>2</td>
<td>Superheterodyne receiver</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Abstraction made by the SDR software</td>
<td>25</td>
</tr>
<tr>
<td>4</td>
<td>Typical schematic of a SDR device</td>
<td>25</td>
</tr>
<tr>
<td>5</td>
<td>Example of NOAA Weather Satellite Images</td>
<td>27</td>
</tr>
<tr>
<td>6</td>
<td>Python Shell</td>
<td>30</td>
</tr>
<tr>
<td>7</td>
<td>Printing Hello World using the Python shell</td>
<td>30</td>
</tr>
<tr>
<td>8</td>
<td>Implicit casting of types</td>
<td>31</td>
</tr>
<tr>
<td>9</td>
<td>Using floating-point numbers</td>
<td>31</td>
</tr>
<tr>
<td>10</td>
<td>Function available for the module math</td>
<td>32</td>
</tr>
<tr>
<td>11</td>
<td>Function math.pow takes two arguments</td>
<td>33</td>
</tr>
<tr>
<td>12</td>
<td>Function pow takes three arguments</td>
<td>34</td>
</tr>
<tr>
<td>13</td>
<td>Evaluating an expression</td>
<td>34</td>
</tr>
<tr>
<td>14</td>
<td>Lists in Python</td>
<td>35</td>
</tr>
<tr>
<td>15</td>
<td>Functions in Python</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>Parameters are modified in the scope of the function</td>
<td>36</td>
</tr>
<tr>
<td>17</td>
<td>How to launch GNU Radio Companion</td>
<td>37</td>
</tr>
<tr>
<td>18</td>
<td>List of blocks, organized by categories</td>
<td>38</td>
</tr>
<tr>
<td>19</td>
<td>Two different signals will be reproduced by the sound card</td>
<td>38</td>
</tr>
<tr>
<td>20</td>
<td>A signal source was added to the canvas</td>
<td>39</td>
</tr>
<tr>
<td>21</td>
<td>Every category is a drop-down list</td>
<td>39</td>
</tr>
<tr>
<td>22</td>
<td>Changing the number of inputs</td>
<td>40</td>
</tr>
<tr>
<td>23</td>
<td>The signal sources are connected to the audio sink</td>
<td>41</td>
</tr>
<tr>
<td>24</td>
<td>Information of the error</td>
<td>41</td>
</tr>
<tr>
<td>25</td>
<td>The output type was changed to float</td>
<td>42</td>
</tr>
<tr>
<td>26</td>
<td>There is not any error in the connections</td>
<td>43</td>
</tr>
<tr>
<td>27</td>
<td>Files generated by GNU Radio Companion</td>
<td>44</td>
</tr>
<tr>
<td>28</td>
<td>Generate options</td>
<td>44</td>
</tr>
<tr>
<td>29</td>
<td>Plain-text editor</td>
<td>45</td>
</tr>
<tr>
<td>30</td>
<td>Source code of the project</td>
<td>46</td>
</tr>
<tr>
<td>31</td>
<td>How to execute the Python script</td>
<td>46</td>
</tr>
</tbody>
</table>
Image 32 - Modules imported in this Project .................................................. 47
Image 33 - Class of the project ................................................................. 47
Image 34 - The variable for the sample rate ................................................ 47
Image 35 - Documentation of the signal source block ............................... 48
Image 36 - Help available for gr_modtool .................................................. 51
Image 37 - There is a Python folder in the main folder of the project ......... 53
Image 38 - Simplified code of the project .................................................. 53
Image 39 - Constructor of the block ......................................................... 54
Image 40 - Parent class and some basic arguments ................................... 54
Image 41 - Vectors to manage the signals ................................................. 54
Image 42 - The function "work" will do all the operations .......................... 55
Image 43 - The output vector is equal to the input vector multiplied by a constant 55
Image 44 - The XML file is inside of the grc folder ................................. 55
Image 45 - Properties of the project, defined in the XML .......................... 55
Image 46 - Arguments, input and output of the out-of-tree block .............. 56
Image 47 - Creating a new folder ............................................................. 56
Image 48 - Compiling the block using make and cmake .......................... 57
Image 49 - The command "sudo make install" will place the files in the proper folders of the system ................................................................. 57
Image 50 - The block is available in the list of blocks and can be placed in the canvas ................................................................. 58
Image 51 - How to place two blocks to plot the function before and after the multiplication ................................................................. 58
Image 52 - Sum of signals ........................................................................ 59
Image 53 - Typical schematic of a SDR peripheral .................................... 60
Image 54 - The SDR peripheral and the sound card will generate digital signals overall ............................................................................. 61
Image 55 - The Generate Options should be changed to WX GUI ........... 62
Image 56 - Values of the receiver block .................................................... 63
Image 57 - There are two variables, samp_rate and my_freq .................. 63
Image 58 - Values for the slider ............................................................... 64
Image 59 - Typical diagram of a superheterodyne receiver ....................... 64
Image 60 - Multiplying the local oscillator with the input signal .............. 65
Image 61 - A filter was added to the design ............................................................ 65
Image 62 - Properties of the filter ........................................................................ 66
Image 63 - Properties of the rational resampler ..................................................... 66
Image 64 – Connections of the rational resampler ............................................... 67
Image 65 - Using an audio sink and a wave file sink .............................................. 68
Image 66 - A FFT sink to visualize the signal received by the SDR peripheral ..... 69
Image 67 - Compilation of the Varicode block ..................................................... 71
Image 68 - Installation of the Varicode block ....................................................... 72
Image 69 - The Varicode blocks will be under the Ham category ....................... 72
Image 70 - How to import a module using GNU Radio Companion ................. 73
Image 71 - Variables used for the transmitter ..................................................... 73
Image 72 - Location of the input file ...................................................................... 74
Image 73 - Connection between the blocks ......................................................... 74
Image 74 - Process of converting an analog signal into a digital signal ............. 75
Image 75 - Setting the symbol table ................................................................. 75
Image 76 - Connection between chunks to symbols and the Interpolated FIR ...... 76
Image 77 - Resampling and multiplication with a constant ................................... 77
Image 78 - Properties of the Osmocom sink block .............................................. 77
Image 79 - The full design of the transmitter ..................................................... 78
Image 80 - Variables used for the receiver .......................................................... 78
Image 81 - Properties of the Osmocom source block .......................................... 79
Image 82 - Superheterodyne receiver ................................................................. 79
Image 83 - Multiplication between the signal source and the Osmocom Source .... 80
Image 84 - Properties of the filter ................................................................. 80
Image 85 - The mentioned blocks ................................................................. 81
Image 86 - Properties of the clock recovery block ............................................ 82
Image 87 - Using the Varicode decoder ............................................................... 82
Image 88 - Either saving or displaying the text ................................................... 82
Image 89 - The whole design of the receiver ..................................................... 83
Adrian Crespo

*Digital transmission and reception techniques with Software Defined Radio*
INDEX

1 Introduction ................................................................................................... 17
  1.1 Purpose of the project .............................................................................. 17
  1.2 Tools used for this project ....................................................................... 18
    1.2.1 New Linux distribution ........................................................................ 18
    1.2.2 SDR Devices ....................................................................................... 21
  1.3 Goal of the project ................................................................................... 23

2 Software Defined Radio ................................................................................ 24
  2.1 Typical circuits with Software Defined Radio ........................................ 24
  2.2 Some typical applications ....................................................................... 26
    2.2.1 Military ................................................................................................ 26
    2.2.2 Radio Astronomy ................................................................................. 26
    2.2.3 Tracking of ships and aircrafts ............................................................ 26
    2.2.4 GSM Networks .................................................................................... 27
    2.2.5 Processing satellite images .................................................................. 27

3 Laboratory exercises with GNU Radio ........................................................ 28
  3.1 Why to use GNU Radio ........................................................................... 28
  3.2 General assumptions ................................................................................ 28
  3.3 INTRODUCTION TO PYTHON ................................................................... 29
    3.3.1 Introduction .......................................................................................... 29
    3.3.2 Hello World ........................................................................................ 30
    3.3.3 Mathematical operations .................................................................... 31
    3.3.4 Variables ............................................................................................. 34
    3.3.5 Functions ............................................................................................. 36
  3.4 Using GNU Radio companion ................................................................... 37
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4.1</td>
<td>Introduction</td>
<td>37</td>
</tr>
<tr>
<td>3.4.2</td>
<td>GNU Radio Companion</td>
<td>37</td>
</tr>
<tr>
<td>3.5</td>
<td>Using Python for GNU Radio</td>
<td>44</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Introduction</td>
<td>44</td>
</tr>
<tr>
<td>3.5.2</td>
<td>Examining the code</td>
<td>46</td>
</tr>
<tr>
<td>3.6</td>
<td>OUT-OF-TREE MODULE</td>
<td>50</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Introduction</td>
<td>50</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Creating a new block</td>
<td>50</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Defining the constructor</td>
<td>54</td>
</tr>
<tr>
<td>3.6.4</td>
<td>Creating the function</td>
<td>55</td>
</tr>
<tr>
<td>3.6.5</td>
<td>Editing the XML configuration</td>
<td>55</td>
</tr>
<tr>
<td>3.6.6</td>
<td>Compiling and installing the module</td>
<td>56</td>
</tr>
<tr>
<td>3.7</td>
<td>FM Reception</td>
<td>60</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Introduction</td>
<td>60</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Receiver</td>
<td>60</td>
</tr>
<tr>
<td>3.8</td>
<td>ASK Transmission and reception</td>
<td>71</td>
</tr>
<tr>
<td>3.8.1</td>
<td>Introduction</td>
<td>71</td>
</tr>
<tr>
<td>3.8.2</td>
<td>Installing the Varicode encoder-decoder</td>
<td>71</td>
</tr>
<tr>
<td>3.8.3</td>
<td>Designing the transmitter</td>
<td>73</td>
</tr>
<tr>
<td>3.8.4</td>
<td>Designing the receiver</td>
<td>78</td>
</tr>
<tr>
<td>4</td>
<td>Conclusions</td>
<td>85</td>
</tr>
<tr>
<td>5</td>
<td>References</td>
<td>87</td>
</tr>
</tbody>
</table>
1 Introduction

1.1 Purpose of the project

The aim of this project is to develop a set of exercises using GNU Radio that could let the students to understand the basics of digital transmission and reception techniques. GNU Radio is a development toolkit based on Python.

From the perspective of a student, a practical approach with real examples may help to support the content of other subjects, such as Signal and Systems, Signal Processing, Modulation and Coding, and Wireless Communications.

From the point of view of an engineer, GNU Radio is not only a fast way to develop new radio protocols, but also is flexible since it can be used with universal software devices, and the cost of reconfiguring and maintaining a system is cheaper because all the changes would be made using software and not relying on hardware. This is the main advantage of replacing analog signal processing with digital signal processing. To point out some disadvantages, this technique requires higher power consumption than an approach based on an application-specific integrated circuit. Also, it uses more millions of instructions per second since it relies on the computer’s processor speed.

In order to make it easy to follow, the exercises have been developed step by step, taking care of any specific concept that could need clarification or that could provide additional information for the student. For instance, if the conditions of a problem may change substantially by modifying the properties of a block or a specific signal, the exercise encourages the student to study it.
1.2 Tools used for this project

1.2.1 New Linux distribution

For the development of the exercises and to provide the students a ready-to-use environment, it was developed a Linux distribution based on Ubuntu containing all the essential tools to work with GNU Radio. Some of its features it that this distribution uses a LXDE desktop environment instead of a GNOME or Unity desktop environment. This decision was taken considering that LXDE requires less RAM memory and less usage of CPU to work (E. Craig, 2014).

![Desktop displaying the default wallpaper](image1.jpg)

*Image 1 - Desktop displaying the default wallpaper*

The first step was to download an Ubuntu Mini ISO image and use the Ubuntu Customization Kit tool to setup the new distribution (Ubuntu Customization Kit main remastering backend script, 2016).

After that, it was required to setup the environment and unpack the ISO image:

```
SUDO UCK-REMASTER-CLEAN
```
Once it was unpacked, the next step was to change the apparent root directory for the current running process by the location where the ISO image was unpacked.

`SUDO UCK-REMASTER-UNPACK-ROOTFS && SUDO UCK-REMASTER-CHROOT-ROOTFS`

Then, to install new software it was required to edit the sources of the packaged that can be found by the Advanced Packaging Tool. It was required to add the “universe” repository to the sources list.

`/ETC/APT/SOURCES.LIST`

In order to install the LXDE Desktop Environment it was possible to install the core already available for Lubuntu:

`SUDO -I`

`APT-GET INSTALL LUBUNTU-CORE`

`APT-GET DIST-UPGRADE`

`APT-GET AUTOCLEAN`

`RM /VAR/CACHE/APT/ARCHIVES/*.DEB`

`REBOOT`
After this was done, it was possible to install all the required software for the environment that will be used by the students to develop any project using GNU Radio. Some of the packages currently installed are:

<table>
<thead>
<tr>
<th>Software installed</th>
<th>gstreamer0.10-plugins-ugly</th>
<th>arj</th>
<th>gstreamer0.10-plugins-ugly-multiverse</th>
<th>cabextract</th>
<th>gstreamer0.10-pitfdll</th>
<th>file-roller</th>
<th>p7zip-rar</th>
<th>Atom</th>
<th>p7zip-full</th>
<th>liborc-0.4.0</th>
<th>unace</th>
<th>liborc-0.4-dev</th>
<th>p7zip-full</th>
<th>libasound2-dev</th>
<th>icewm</th>
<th>libzmq1</th>
<th>python-gtk2</th>
<th>unzi</th>
<th>libzmq1-dev</th>
<th>rar</th>
<th>libboost-all-dev</th>
<th>uudeview</th>
<th>ubuntu-restricted-extras</th>
<th>mpack</th>
</tr>
</thead>
<tbody>
<tr>
<td>testdisk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reiserfsprogs</td>
<td>gstreamer0.10-plugins-ugly-multiverse</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ntfs-3g</td>
<td>gstreamer0.10-pitfdll</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dosfstools</td>
<td>p7zip-rar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>casper(alreadyinstalled)</td>
<td>p7zip-full</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ubiquity</td>
<td>unace</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ubiquity-frontend-gtk,</td>
<td>unrar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>icewm</td>
<td>zip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>icewm-themes</td>
<td>unzip</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>faad</td>
<td>sharutils</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gstreamer0.10-ffmpeg</td>
<td>rar</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gstreamer0.10-plugins-bad</td>
<td>uudeview</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gstreamer0.10-plugins-bad-multiverse</td>
<td>mpack</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 1 - Software installed in the Linux distribution*

This software was installed using the Advanced Package Manager (APT), for instance:

*SUDO APT-GET INSTALL ZIP*
The installation of GNU Radio was made using a script that handles all the dependencies and requirements needed to install the software:

\textit{wget http://www.sbrac.org/files/build-gnuradio}

\textit{sudo chmod a+x ./build-gnuradio}

\textit{sudo ./build-gnuradio}

Eventually, it was possible to customize the desktop through editing the file:

\texttt{/home/student/.config/pcfman/}

\textbf{The user by default was “student” and the password was also “student”.}

\subsection{1.2.2 SDR Devices}

The exercises were developed initially using a Digital Video Broadband receiver. This receiver had a chipset RTL2832U and a tuner R820T2. The frequency range was between 24 and 1766 MHz, 3.2 MHz of bandwidth, and the antenna was a telescopic antenna of 6 cm of length.

\begin{table}[h]
\centering
\begin{tabular}{|l|l|}
\hline
\textbf{RTL-SDR Tuner} & \textbf{Frequency Range} \\
\hline
Elonics E4000 (E4K) & 54 - 2200 MHz \\
Rafael Micro R820T & 24 - 1766 MHz \\
Rafael Micro R820T2 & 24 - 1766 MHz \\
Fitipower FC0013 & 22 - 1100 MHz \\
Fitipower FC0012 & 22 - 948 MHz \\
FCI FC2580 & 146 - 308 MHz and 438 - 924 MHz \\
\hline
\end{tabular}
\caption{Comparison of RTL-SDR Tuners}
\end{table}

Since the manufacturer Elonics ceased to operate some years ago, it is very difficult to find that kind of tuners and the best option was to use the R820T2. However, despite it has not been proved yet, it seems that those tuners have a bad performance when the frequency is higher than 1500 MHz without cooling.
Because of the limited range of frequency and the low performance of the Digital Video Broadband receivers, the exercise was eventually developed to be used with a HackRF peripheral. This peripheral is capable of reception and transmission of signals between 30 MHz and 6 GHz with a bandwidth of 20 MHz.

This is a list of the features for the some of the most used SDR peripherals currently (Killian, 2013):

<table>
<thead>
<tr>
<th>Feature</th>
<th>HackRF</th>
<th>bladeRF</th>
<th>USRP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B100 Starter</td>
<td>B200 Starter</td>
<td>B210</td>
</tr>
<tr>
<td></td>
<td>x40</td>
<td>x115</td>
<td></td>
</tr>
<tr>
<td>Radio Spectrum</td>
<td>30 MHz – 6 GHz</td>
<td>300 MHz – 3.8 GHz</td>
<td>50 MHz – 2.2 GHz</td>
</tr>
<tr>
<td>Bandwidth</td>
<td>20 MHz</td>
<td>28 MHz</td>
<td>16 MHz</td>
</tr>
<tr>
<td>Duplex</td>
<td>Half</td>
<td>Full</td>
<td>Full</td>
</tr>
<tr>
<td>Sample Size (ADC/DAC)</td>
<td>8 bit</td>
<td>12 bit</td>
<td>12 bit</td>
</tr>
<tr>
<td>Sample Rate (ADC/DAC)</td>
<td>20 Msps</td>
<td>40 Msps</td>
<td>64 Msps / 128 Msps</td>
</tr>
<tr>
<td>Interface (Speed)</td>
<td>USB 2 HS (480 megabit)</td>
<td>USB 3 (5 gigabit)</td>
<td>USB 2 HS (480 megabit)</td>
</tr>
<tr>
<td>FPGA Logic Elements</td>
<td>[4]</td>
<td>40k</td>
<td>25k</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>LPC43XX</td>
<td>Cypress FX3</td>
<td>Cypress FX2</td>
</tr>
<tr>
<td>Open Source</td>
<td>Everything</td>
<td>HDL + Code</td>
<td>HDL + Code</td>
</tr>
<tr>
<td></td>
<td>Schematics</td>
<td>Schematics</td>
<td>Schematics</td>
</tr>
<tr>
<td>Cost</td>
<td>$300</td>
<td>$420</td>
<td>$650</td>
</tr>
</tbody>
</table>

Table 3 - Comparison of SDR peripherals
It should be noticed that the HackRF is not capable of full duplex communication, but anyways transmitting and receiving at the same time will cause a lot of noise on the receiver for every board. Because of that, the best option was to use two different HackRF peripherals, one for transmission and another one for reception.

A disadvantage of using HackRF is that it has a USB 2.0 port, which limits the peak of data transmission speed to about 35 MBytes/second. If it is used a Linux distribution running on a Live USB drive, there will be a limit imposed by the operative system and another one for the SDR peripheral. The exercises were developed using this technique and there was not any noticeable delay or problem with the performance, in general, using a Live USB with USB 3.0.

However, if instead of using a Live USB drive it is used a virtual machine, the speed will decrease noticeably. Despite the speed was not measured, the reception and transmission of data was so poor that it barely worked.

### 1.3 Goal of the project

The exercises elaborated for this project are intended to let the students have a practical approach to the concepts studied in some subjects such as Digital Signal Processing or Modulation and Coding. Currently, some laboratories are using Matlab to analyze and study different signals and systems, but using Software Defined Radio provide a step-by-step guidance about how to apply the theory to reach an effect on real environments.

Specifically, the aim of the project is to support the following concepts:

- To understand of some transmission and reception techniques.
- To understand of some basic concepts of signals and systems, such as the sample ratio, frequency allocation, noise filtering, window functions, filter design, clock recovery, codification, etcetera.
- To apply a modern high-level programming language to transmission and reception systems.
- To test and measure the effect of different parameters and characteristics to real cases-
- To be familiar with some basic designs and circuits, such as the superheterodyne receiver or some basic demodulation processes.
Also, this framework may be useful for the students since there seems to be an effort to replace some hardware components by software because of some benefits such as the code reuse and in-service upgrades (SDR Forum, s.f.). Therefore, by using GNU Radio they may be learning some new skills that could be highly-demanded in a near future.

2 Software Defined Radio

2.1 Typical circuits with Software Defined Radio

Typically, to receive a signal is required a suitable antenna. The output of the antenna could be very small, so it usually needs to be amplified using a radio frequency amplifier. In order to tune the receiver, it will be mixed the original signal with a sinusoidal signal produced by a local oscillator. Because of this, there may be high-order intermodulation products that should be filtered out in an intermediate frequency amplifier (IF Amplifier). The received signal is then processed by a demodulator stage where the audio signal is recovered, amplified and processed.

![Image 2 - Superheterodyne receiver](image)

Using Software Defined Radio it is possible to implement some software capable of handle the operations made by the radio frequency amplifier, the intermediate frequency amplifier and the demodulator.
But what is the purpose of the Software Defined Radio peripheral? Typically, all devices are a quadrature sampler detector. They have an antenna and they are able to obtain the imaginary part and the real part of a signal received by the antenna. The frequency of what part of the RF spectrum is demodulated can be changed by the variable frequency oscillator (VFO).

The output of these devices will be two digital signals containing information about the in-phase and quadrature components. From the point of view of a student, this can be black boxed since there is a sound card in the computer where the SDR device is connected. This sound card will handle the I/Q components and will transform the signal from analog to digital form.
2.2 Some typical applications

The Software Defined Radio techniques can be used in different projects involved in processing and analyzing signals. Below, some projects that are being developed using Software Defined Radio will be presented.

2.2.1 Military

Some military applications required a flexible new approach to meet diverse soldier communications needs. Because of that the US Army, with the collaboration of some manufacturers such as Thales and General Dynamics Mission Systems, created a program to developed a new communications architecture through the use of Software Defined Radio (Feickert, 2005).

2.2.2 Radio Astronomy

In the 1930s, the emission of radio waves from celestial objects was observed accidentally by Karl Jansk while he was studying different sources of static on international radio-telephone circuits. As a recognition for his efforts, the fundamental particle of flux density was named after him, the jansky (Jy).

Nowadays, the Software Defined Radio is playing a major role in Radio Astronomy by replacing some applications done with application specific integrated circuits (ASIC). For example, there is a project which uses a low noise amplifier a satellite dish to analyze some properties of the Sun and the galactic plane in continuum and spectral modes. This project was able to get data to analyse the Galactic Plane and the Hydrogen Line (Leech, 2012). The Hydrogen Line is usually used to calculate the rotation curve of a galaxy.

2.2.3 Tracking of ships and aircrafts

Some Air Traffic Control techniques rely on Software Defined Radio. The primary purpose is to organize the flow of air traffic and to prevent collisions. Through the Automatic Identification System, it is possible to use Software Defined Radio applications as a supplement to marine radar to locate and identify ships. Also, some aircrafts can be located through different modes of interrogation, e.g. the Mode Select of the aviation transponder interrogation modes (RTL-SDR Tutorial: Cheap ADS-B Aircraft RADAR, 2013).
2.2.4 GSM Networks

Some Software Defined Radio applications can analyze cellular phone GSM signals or even implement a GSM network using some software implementations such as OsmoBTS or OpenBTS. They can be used to create an implementation which may interact with the traditional GSM network by using the Abis protocol, but also it may be used for Voice over IP by using an Asterix, an open source telephony switching (Androulidakis, 2012).

2.2.5 Processing satellite images

It is possible to develop Software Defined Radio applications to receive and process different signals sent from satellites. In 2015, some plugins were developed to receive QPSK modulations and process Meteor-M2 LRPT images (Beliakov, 2015). In a similar way, it can be used to process signals sent to Earth by NOAA satellites (RTL-SDR Tutorial: Receiving NOAA Weather Satellite Images, 2013).

3 Laboratory exercises with GNU Radio

3.1 Why to use GNU Radio

GNU Radio is a free software released under the GNU General Public License with more than 15 years of development. Because of that, it is a consolidated toolkit that provides capabilities to implement signal processing systems.

GNU Radio has a graphical user interface called GNU Radio Companion. This is the front/end used to design signal processing systems graphically. Despite GNU Radio can be use with C++, GNU Radio Companion only generates the code in Python, that is why Python is the most used language and that is why the laboratory will be based on Python.

There are some other frameworks, such as Matlab or LabView, that can be used with the USRP hardware driver provided by Ettus Research. Why GNU Radio was chosen instead of other frameworks?

LabView usually requires that all the plugins need to be developed with C++ and C. Despite GNU Radio Companion only generate Python code, the libraries of GNU Radio can be used with either Python or C and C++. Also, using an interpreted language provides some advantages, for example any modification can be done easier and faster.

Also, Matlab and LabView are released under proprietary license. Instead of that, GNU Radio is released under the GNU General Public License, which makes GNU Radio open source. This makes GNU Radio a cheaper alternative to Matlab and LabView.

3.2 General assumptions

This project requires some basics of programing in Python and knowledge of signals and systems as well as digital signal processing. However, in order to help the students there will be a short introduction to the Python language programming.

The exercises require to know some concepts of signal classification: discrete time systems and their description; deterministic, continuous time and elementary signals; as well as the orthogonal signals and orthonormal signal bases.
Also, it would require some knowledge of systems and filters: classification of linear and nonlinear systems, time-invariance, system response to elementary excitation; linear and circular convolution; filtering; connecting systems in parallel and in chain.

It would need some mathematical concepts such as the Z transform, its properties or the relation between the Z transform and Discrete Time Fourier Transform.

At last, the student should know some basics of sampling: what are band-limited signal, the sampling theorem, conversion between analog and digital signals and vice versa and the signal to noise ratio.

3.3 INTRODUCTION TO PYTHON

3.3.1 Introduction

Python is an interpreted, object-oriented, and high-level programming language. It was created during 1985-1990 by Guido van Rossum and It is delivered under the important differences in this language that must be considered when using Python to develop projects for GNU Radio:

- Types: In some languages the types are strictly declared. This is the case of C++, the type must be defined before using the variable. In Python, the types are assigned during runtime and they cannot be easily changed.
- All parameters are passed by value in Python. All variables are references to objects, this means that when you do a copy of a variable, you are doing another reference to the same objects. In C, it is similar to getting a copy of a pointer.
- Memory management: C++ does not have garbage collections while Python does. For example, Python maintains a reference count for every reference made to an object: when the reference is zero, it deallocates the object because it is no longer used. Beyond this technique, it uses more specific algorithms to enhance the performance of the memory.
- Interpreted language: While other languages are compiled, Python is an interpreted language that does not need to run a compilation stage.
3.3.2 Hello World

Though typically synonymous with a shell or text terminal, a terminal allows the user access to all its applications such as the command line interfaces. Python uses a command line interface; to use the Python interpreter it is needed to use the shell. The terminal could be the program XTerm from the menu of applications.

At the shell prompt, the following command launches the Python shell:

![Image 6 - Python Shell](image)

This is the interpreter, it uses Python functions and it can evaluate expressions. For example, it is possible to print strings with the print command:

![Image 7 - Printing Hello World using the Python shell](image)
Questions

a) Print a different message in the screen using the Python shell.

3.3.3 Mathematical operations

Python is able to evaluate mathematical expressions using the interpreter. If only integers are used, then the results will be rounded:

![Image 8 - Implicit casting of types]

To use a floating-point variable, it needs to be specified a decimal point, at least in one of the operands. Python will manage the types to generate a result in decimal format:

![Image 9 - Using floating-point numbers]
To use more advanced operations, the interpreter needs to load the mathematical module. To load a module, it is needed to use the “import” command. To list what is available in this module, use the command `dir(math):

![Image](image.jpg)

**Image 10 - Function available for the module math**

These are the functions that belong to the module math. To get information about how to use any function from the mathematical module, it is possible to check the help information for any command:

`help(math.name)`
In this example, it is shown how to use the function `pow`.

![Image](image1.png)

*Image 11 - Function `math.pow` takes two arguments*

*Read the help for the function `sqrt`.*

However, it should be noticed that it is not the same function if the module name is omitted. For example, the built-in function `pow`:

```
help(pow)
```
Surprisingly, it is a different function that accepts up to three arguments. It is shown that the third argument is used to calculate the module, which can be useful when performing calculations in modular arithmetic.

![Image 12](image.png)

**Image 12 - Function pow takes three arguments**

### 3.3.4 Variables

These mathematical expressions generate a result that can be assigned to a variable. While in Matlab you can specify if you want to display the output of an operation, by default it is omitted in Python, so that to display the result, you must either evaluate the variable or print it:

![Image 13](image.png)

**Image 13 - Evaluating an expression**
Lists are denoted by square brackets and you can assign a list to a variable. Also, they can have any kind of value inside, for example strings and integers:

```
student@ubuntu:~$ python
Python 2.7.6 (default, Jun 22 2015, 17:58:13)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> my_list = []
>>> my_list = [1,2,3,4]
>>> my_list
[1, 2, 3, 4]
>>> my_list = [1,2,“loz”,4]
>>> my_list
[1, 2, ‘loz’, 4]
```

*Image 14 - Lists in Python*

To select an element from a list, use square brackets. As it happens in C++, the first element of the list will have an index zero. Also, it can be specified a range of values just like in Matlab:

```
\[
\frac{3^2}{4} \log_{10} 10 \\
\cos(\arctan(1)) \\
\sqrt{2} \sin \pi \\
x \leftarrow 3; y \leftarrow 3; xy \\
x \leftarrow \cos(\frac{\pi}{4}); y \leftarrow \ln(100); \frac{x + y}{y}
\]
```

*Consider the following expressions in Python and evaluate them:*

```
\[
3^2 \log_{10} 10 \\
\cos(\arctan(1)) \\
\sqrt{2} \sin \pi \\
x \leftarrow 3; y \leftarrow 3; xy \\
x \leftarrow \cos(\frac{\pi}{4}); y \leftarrow \ln(100); \frac{x + y}{y}
```

---

35
3.3.5 Functions

In order to define functions, the block of code begins with the keyword def followed by the function name and parentheses ( ( ) ), and inside of them there can be arguments.

The following example takes an argument and prints the argument as a string. In Python there is no need to specify the type of the arguments:

```
In order to define functions, the block of code begins with the keyword def followed by the function name and parentheses ( ( ) ), and inside of them there can be arguments.

The following example takes an argument and prints the argument as a string. In Python there is no need to specify the type of the arguments:

```

All parameters (arguments) in the Python language are passed by reference. In this example the following does not need to return the list to modify its value, since it is passed by reference:

```
```

create a list containing the following elements \([3, 4, 6, 1, 5, 8]\) and define a function to pow to the second power every element of the list.

```

create a list containing the following elements \([3, 4, 6, 1, 5, 8]\) and define a function to pow to the second power every element of the list.

```

Questions

Create a list containing the following elements \([3, 4, 6, 1, 5, 8]\) and define a function to pow to the second power every element of the list.
3.4 Using GNU Radio companion

3.4.1 Introduction

This laboratory will explain the basics of GNU Radio, how do the GRC files work and how to create our own Python code to develop a real project. It requires to be familiar with Python because despite there is a graphical interface to design systems, it could be needed to create a block or to add some functionalities that are not easily reachable with the graphical interface.

GNU Radio is a free software released under the GNU General Public License with more than 15 years of development. Because of that, it is a consolidated toolkit that provides capabilities to implement signal processing systems.

3.4.2 GNU Radio Companion

GNU Radio has a graphical user interface called GNU Radio Companion. This is the front/end used to design signal processing systems graphically. Despite GNU Radio can be use with C++, GNU Radio Companion only generates the code in Python, that is why Python is the most used language and that is why the laboratory will be based on Python.

GNU Radio Companion can be opened by typing its name in a terminal:

![Image 17 - How to launch GNU Radio Companion](Image17.png)
It will open a GNU Radio Companion which will show a blank canvas with two default blocks, one of them for the options and another one will be a variable for the sampling rate of the system. The list is displayed at the right and it will let to add blocks in this canvas.

![List of blocks, organized by categories](image18.png)

The purpose of this exercise is just to generate two sines and to send them to the sound card of the system so they will be reproduced in the speaker:

![Two different signals will be reproduced by the sound card](image19.png)
Following the convention, the input of the system will be the **source** while the output will be the **sink**. In this case, the signal source must be connected to a sink to generate an output. For this example, we will connect two sources to an audio sink.

By double-clicking in any element from that list, it will be added directly to the canvas. The sine generator can be found in the **Signal Source** label, under the **Waveform Generators** category. It will have a red name, which indicates that it needs to be connected somewhere in order to work.

![Image 20 - A signal source was added to the canvas](image-url)

It would be needed to add another signal source and an audio sink, which could be found under the **Audio** category:

```
- [Audio]
  Alaw Audio Decoder
  Audio Sink
  Audio Source
  CODEC2 Audio Decoder
  CODEC2 Audio Encoder
```

![Image 21 - Every category is a drop-down list](image-url)
The three blocks must be connected. However, it is not possible to connect two sources to the same input directly. The audio sink should have two inputs, but it is possible to change the properties of the audio sink by doing double-click on its block. Then, the number of inputs can be set to two:

![Image 22 - Changing the number of inputs](image.png)
Now, by clicking the blue tab from the signal sources and then clicking in the orange tab from the audio sink it will be possible to connect the blocks.

![Image 23 - The signal sources are connected to the audio sink](image)

There will be two red arrows pointing the audio sink. Any error can be checked by clicking the exclamation mark in the tool-bar. An emerging window will show and explain any error in the Python code that will be generated eventually.

![Image 24 - Information of the error](image)
Adrian Crespo  
Digital transmission and reception techniques with Software Defined Radio

The error warns:

*Source IO size "8" does not match sink IO size "4".*

This error means that the source signal is generating a type of data which is not compatible with the type of data accepted by the next block, in this case by the sink. It is possible to modify the properties of the source signals to change the type. To achieve this, the properties can be accessed by doing double-click to the blocks. Then, the only change that it is needed to solve the problem is to modify the property Output Type in both blocks from Complex to Float:

![Image](25)  
*Image 25 - The output type was changed to float*
Now, the arrows pointing the audio sink will be black and it will be possible to test the system. There is an action in the tool-bar, under the Tools section, to execute the project:

Image 26 - There is not any error in the connections

It will ask to save the project and afterwards it will compile and run the project.

Questions

a) What happens if the property for the frequency of the two sources changes?

b) Change the waveform. Instead of a cosine, what happens if it is a constant?

c) How it would be possible to modify the amplitude to 0.5?

d) What would happen if the sample rate is half of the sample rate and how is related to the sample rate of the audio sink?
3.5 Using Python for GNU Radio

3.5.1 Introduction

In the previous example, two audio sources were connected to an audio sink using GNU Radio Companion. GNU Radio Companion generated two files in the same path where the project was saved:

![Image 27 - Files generated by GNU Radio Companion](image)

The **GRC file** is a XML file containing the configuration and the design of the project. This file includes the position of the blocks, the value of the variables or some specific parameters for the environment. The GRC file is used by the GNU Radio Companion to generate the second file, the Python file located in the same folder.

The name of this Python file is **top_block.py because it was the ID of the project**, which could be changed in the Options block by doing double click in the block and changing the property.

![Image 28 - Generate options](image)
It is worth to mention the property **Generate Options**, which by default is **QT Gui**. Actually, it defines on which graphical library it will be based on, **QT or WX**. For each one we will have slightly different blocks, mostly those who are used for plotting, but actually the main blocks and functions are the same. A **common error** is to use a **WX GUI FFT Sink** block when the Generate Options is set to QT, this error will prevent the project to be compiled.

The Python file generated by GRC contains the Python code generated from the graphical design. If can be opened with a plain-text editor, for example Atom.

---

![Image 29 – Plain-text editor](image)

Once it is opened, it will display the whole code of the project:
3.5.2 Examining the code

The first line that can be found in the Python file is:

```
#!/usr/bin/env Python
```

This line indicates to the Linux console that it is a Python file and how should it use it. This means that the file could be executed using the Python interpreter through the terminal, without using the GNU Radio Companion. This Python script can be executed in two analog ways:

```
student@ubuntu:~/ZTRadio/Ch2  ─ + ×
student@ubuntu:~/ZTRadio/Ch2$ dir
ch2,dve,ch2_part2,dve,ch2_part2.dve,bak" sines.gro
student@ubuntu:~/ZTRadio/Ch2$ python top_block.py
INFO: Audio sink arch:alsa
student@ubuntu:~/ZTRadio/Ch2$ ./top_block.py
INFO: Audio sink arch:alsa
student@ubuntu:~/ZTRadio/Ch2$ [ ▼ ]
```

This Python script could be executed in a system without any graphical interface, for example in a remote server, an Arduino or a Raspberry Pi. There are also other advantages of running directly a Python script: for example, a system could be configured to run it periodically using the Cron service to analyze the data from a weather satellite.

Going deeper in the analyses of the code, there are some lines to load the modules for the project in the Python script. The **module gr must be present** always to run the project. The audio sink is included in the audio module and the signal source is included in the analog module:
Questions

a) How could you load the module math? Is there any function to set the amplitude of the signals to the square root of 3?

After that, there is a definition for the class of the project, which will have the same name as the ID of the project that was defined in GNU Radio Companion:

```python
class top_block(gr.top_block, Qt.QWidget):
```

The following lines can be ignored until the Variables section. The only variable in the project was the sample rate, which ID was samp_rate. It is set to 32000, so it means that by default that will be its value, unless something changes it:

```python
# Variables
self.samp_rate = samp_rate = 32000
```
a) Define a variable for the frequency of the signals. Which argument is the frequency in the example?

The following section is the Blocks section, that will define the blocks of our system. The manual of GNU Radio provides all the information to know which arguments accept each function:

Manual:
http://gnuradio.org/doc/doxygen/classgr_1_1analog_1_1sig__source__f.html

For the signal source block, the documentation specifies the following arguments in the Member Function Documentation section:

**Build a signal source block.**

**Parameters**

- `sampling_freq` Sampling rate of signal.
- `waveform` Waveform type.
- `wave_freq` Frequency of waveform (relative to sampling_freq).
- `ampl` Signal amplitude.
- `offset` Offset of signal.

Notice that the function takes five parameters but in this example there are used only four. This is not wrong, since the last parameter, the offset, is set to 0 by default. This will not happen with any other argument which has not a value set by default, in that case it is required to provide a value in order to build and execute the project.

The last section of the source code is the Connections section.

```python
# Connections
self.connect((self.analog_sig_source_x 0, 0), (self.audio_sink_0, 0))
self.connect((self.analog_sig_source_x 1, 0), (self.audio_sink_0, 1))
```
The function connect will link the blocks between them, but to specify which output matches which input, each block needs a value to indicate the number of the port. Due to the audio sink had two ports, the signal source will be connected to the ports 0 and 1 of the audio sink.

The rest of the code is related to the graphical layout of GNU Radio Companion or to initialization of the objects and variables. For the sake of simplicity, they can be ignored. In the example it will be shown that they can be omitted if the project is not intended to be used in GNU Radio Companion.

Questions

Use the example to answer the following questions.

a) Create a third tone and link it to the audio sink too. Do not forget to use a third port for the sink.

b) What are the differences between the functions “sig_source_f” and ”sig_source_c”?

c) Is there any other function to generate noise? Check the documentation and replace one of the three tones by noise.
3.6 OUT-OF-TREE MODULE

3.6.1 Introduction

This exercise will introduce the basic concepts to understand how to create new blocks for GNU Radio and how to install them in GNU Radio Companion. These blocks are useful to simplify the design of a processing system but it can be used to add a new functionality that it is not included by the default blocks or that does not meet the requirements for the system. For example, if some specific filter is not included in the default blocks, it could be easily implemented using Python and imported to GNU Radio Companion.

These blocks are known as "out-of-tree module" and many of these blocks can be found at the Comprehensive GNU Radio Archive Network. It is worth mentioning that those blocks follow some conventions and style-guides, for example they include some naming conventions.

3.6.2 Creating a new block

The script `gr_modtool` simplifies some of the tasks to developed a new block. Usually, those task involve editing configuration files, such as the makefiles, and `gr_modtool` simplifies it by editing them and by using boilerplates. Some of its options can provide more information by using the command help. For instance, “help newmod” explains how to create a new out-of-tree module:
The command needs an argument for the name to create a new module. It will be asked interactively when the following command is executed:

gr_modtool newmod
This module could include one or more blocks. For example, it could have a block for transmission, another one for reception. Because of that, a module could be understood as a project. There are some options that are needed to create a new block:

If they are not specified, they will be asked interactively. There are two options that are relevant, the type of the block (-t) and the programming language that will be used to create the block, in this case Python but it could be also C++. Therefore, in order to add a new block is needed to run the following command:

```
Gr_modtool add
```

The type defines the ratio between the amount of data at the input and the amount of data at the output and set some restrictions for that. For example, an interpolator
will have a relative rate greater than one, while the decimator will have a relative rate less than one. The type “sync” is used when the amount of input data will generate the same amount of output data.

The gr_modtool script will create the basic structure for the block, including the configuration files, the manifests and the templates. The Python code for this new block will be inside of the folder python, while the folder docs will have the documentation of the block and the grc folder will include the XML manifest used by GNU Radio Companion:

![Image 37 - There is a Python folder in the main folder of the project](image.png)

The Python file will contain the code for the block. The name of the block finishes with a “_ff” as a convention, which specifies what type of data will use input data and output.

For the shake of simplicity, the code will be replaced by this one:

![Image 38 - Simplified code of the project](image.png)
3.6.3 **Defining the constructor**

The import function has been already explained in the previous examples, they can be skipped until the init function. It is a constructor that has two arguments: the first one is mandatory, it is a reference to the own object, it is used to have visibility to the whole scope of the object. The second argument will be the integer that will be used to multiply the amplitude of the signal.

```python
8    def __init__(self, multiple):

Image 39 - Constructor of the block
```

The following lines are using the properties **inherited from the parent**. The parent type is `gr.sync_block` and it is a default block of GNU Radio, it is the base type for any block used in GNU Radio. It will need three arguments in this **second constructor**: the name of the block, an input vector and an output vector:

```python
9    gr.sync_block.__init__(self, 
10    name="multiply_py_ff", 
11    in_sig=[<numpy.float32>], 
12    out_sig=[<numpy.float32>])

Image 40 - Parent class and some basic arguments
```

The symbols `<+...+>` mean that it is a placeholder created automatically by `gr_modtool`. Since it will use a vector of float elements, the type should be `numpy.float32`.

```python
11    in_sig=[numpy.float32], 
12    out_sig=[numpy.float32])

Image 41 - Vectors to manage the signals
```

The last line only assigns the argument to the multiple that is stored in the properties of the current class for the object that will be generated during run-time:

```python
13    self.multiple = multiple
```
3.6.4 Creating the function

The function work will have the code corresponding to the operations. It requires again a reference to the own object, self, and it takes two additional arguments: the input and the output of the function. It is worth to remember that all arguments are passed by reference, that is why it is specified the output in the arguments of the function.

```
def work(self, input_items, output_items):
```

*Image 42 - The function "work" will do all the operations*

Eventually, the input vector, stored in a variable with a shorter name just for simplicity, can be multiplied by the multiple and assigned to the output.

```
in0 = input_items[0]
out = output_items[0]
out[:] = in0*self.multiple
return len(output_items[0])
```

*Image 43 - The output vector is equal to the input vector multiplied by a constant*

3.6.5 Editing the XML configuration

Inside of the “grc” folder, there is a XML file. This file is used by GNU Radio Companion to define the properties of the block. There will be one XML file for each block in the folder:

```
student@ubuntu:~/gr-multiplier/grc - + x
student@ubuntu:~/gr-multiplier/grc# ls
Makefile.txt multiplier My_multiplier_ff.xml
```

*Image 44 - The XML file is inside of the grc folder*

Some properties, such as the name or the category of the block, can be edited in the following lines. The key needs to be unique because it will identify the block, if there are two blocks with the same key it will lead into an error:

```
<name>my_multiplier_ff</name>
<key>multiplier My_multiplier_ff</key>
<category>multiplier</category>
<import>import multiplier</import>
<make>multiplier My_multiplier_ff()</make>
```

*Image 45 - Properties of the project, defined in the XML*
The “param” nodes define each argument of the block. In this example there is only one block named “multiple”, but if there were more arguments, there should be one node for each argument of the function. The multiple needs to have the same type that was specified in the Python code: The same will happen with the sink and the source:

```xml
<param>
  <name>Multiple</name>
  <key>multiple</key>
  <type>float</type>
</param>

<sink>
  <name>in</name>
  <type>float</type>
</sink>

<source>
  <name>out</name>
  <type>float</type>
</source>
```

**Image 46 - Arguments, input and output of the out-of-tree block**

### 3.6.6 Compiling and installing the module

In order to compile the module, it is a good practice to create a build folder inside of the main folder of the module. The compiler will generate a lot of files and creating a new folder will keep the rest of the project clean.

```bash
student@ubuntu:~/gr-multiplier/build$ mkdir build
student@ubuntu:~/gr-multiplier$ cd build/
student@ubuntu:~/gr-multiplier/build$ cmake .
student@ubuntu:~/gr-multiplier/build$ make
```

**Image 47 - Creating a new folder**
The project would be compiled by using “cmake” and “make” inside of this folder. These are the tools that will follow some internal steps in order to compile the block:

```
student@ubuntu:...r-multiplier/build  - + ×
student@ubuntu:~/gr-multiplier/build$ cmake .. && make
```

*Image 48 - Compiling the block using make and cmake*

Once it has finished compiling the module, it can be installed using “make install” as an administrator. It will copy the XML and Python files into the proper folder in the system, usually `/usr/local/share/gnuradio/grc/blocks/` for the XML file and `/usr/local/lib/python2.7/dist-packages/` for the Python files.

```
student@ubuntu:...r-multiplier/build  - + ×
usr/local/lib/libgnuradio-pat.so
GNURADIO_RUNTIME_FOUND = TRUE
-- No C++ sources, skipping lib/
-- No C++ sources, skipping swig/
-- Found PythonInterp: /usr/bin/python2 (found suitable version '2.7.6', minimum required is '2')
-- Configuring done
-- Generating done
-- Build files have been written to: /home/student/gr-multiplier/build

Scanning dependencies of target pygen_python_65c84
[ 33%] Generating _init__.pyc, my_multiplier_ff.pyc
[ 63%] Generating _init__.pyc, my_multiplier_ff.pyc
[100%] Built target pygen_python_65c84
Scanning dependencies of target pygen_appx_946dd
[100%] Built target pygen_appx_946dd
Scanning dependencies of target doxygen_target
[100%] Built target doxygen_target

student@ubuntu:~/gr-multiplier/build$ sudo make install
```

*Image 49 - The command "sudo make install" will place the files in the proper folders of the system*
At this point the module will be available in GNU Radio Companion:

**Image 50 - The block is available in the list of blocks and can be placed in the canvas**

The effect can be visualized and compared between the output and the input of this new block u sink to display the time domain of the signal at the input and at the output of the multiplier block.

**Image 51 - How to place two blocks to plot the function before and after the multiplication**
Questions

a) Create a new out-of-tree module with a block that given two vectors, the vector at the output will be the element-wise addition of the two vectors, as shown in the next diagram.

b) Compile and install the block. Then, use it to sum two different signals and send them to an audio sink, as it was done in the previous laboratories.

c) Display the output of the block using a QT GUI Time Sink.

![Diagram of signals being summed](Image 52 - Sum of signals)
3.7 FM Reception

3.7.1 Introduction

The purpose of this exercise is to design a wide band FM receiver using GNU Radio Companion. This design requires to know the specifications of the transmitter and the receiver. It was developed for a HackRF One because it has a better performance than a DVB-T receiver. Therefore, the main features of this device are:

- It has an operating frequency between 1 MHz and 6GHz.
- It accepts up to 20 million samples per second.
- It could work as a half-duplex transceiver.

Since this exercise only works for reception and not for transmission, it complies with the regulations governing transmission of radio signals in Poland.

3.7.2 Receiver

Typically, a Software Defined Radio peripheral has an antenna and it is able to obtain the imaginary part and the real part of a signal received by the antenna. The frequency of what part of the RF spectrum is demodulated can be changed by an internal variable frequency oscillator (VFO).

![Typical schematic of a SDR peripheral](image53.png)

The output of these devices will be two digital signals containing information about the in-phase and quadrature components. From the point of view of a student, this can be black boxed: the computer where the SDR peripheral is connected will have a sound card. This sound card is responsible of handling the in-phase and quadrature components and generating a digital signal.
Because of that, when there is a source with a SDR peripheral, the student should only care about having a digital signal at the output of the source block.

Based on this assumption, this part will cover the design of the receiver, that is the software that will do all the operations on the digital signal.
Firstly, it is needed to setup the project. That can be done by editing the properties of the options block. The id needs to be changed and for this exercise the generate options will be set as WX GUI because of the blocks that will be used.

Then, the first element to add will be the “osmocom Source”. As it was mentioned above, it will produce baseband samples by sampling radio-frequency on its antenna at a particular frequency, sample rate and gain.

- The sample rate and the frequency will depend on variables that could be easily changed during run-time.
- The frequency correlation is zero, because the relationship between the received signal and the reception time will be covered later.
- The bandwidth will be zero.
- The RF gain usually operates between 50 and 70 Ohms termination in its design, while the IF gain usually have between 200 and 600 Ohms. The main difference between those gains is that the IF gain is used typically for crystal, ceramic and mechanical filters. The values can be set as shown in the picture.
Based on these properties, the block can be setup to use the following values:

![Image 56 - Values of the receiver block](image)

After that, there will be two variables, the sample rate and the frequency, that will need their own block. The frequency will be a WX GUI Slider because in this way it will be easy to modify it using a slide during run-time.

![Image 57 - There are two variables, samp_rate and my_freq](image)
For the WX GUI Slider, there will be a minimum value and a maximum value to select frequencies and they can use scientific notation. The number of steps defines how many different values there will be between the maximum and the minimum.

![Image 58 - Values for the slider](image)

The input signal needs to be multiplied by a cosine acting as a local oscillator, as if it were a super-heterodyne receiver. The frequency of the local oscillator is set so the desired reception radio frequency mixes to intermediate frequency. The mixer may inadvertently produce additional frequencies and because of that there should be a filter to remove all but the desired signal.

![Image 59 - Typical diagram of a superheterodyne receiver](image)
To multiply the local oscillator with the input signal, there is a block that will do it by itself, it is found by default in GNU Radio Companion and does not require installation:

![Image 60](image60.png)

*Image 60 - Multiplying the local oscillator with the input signal*

At this point, there should be a filter to reduce the noise and to eliminate any other signals or their harmonics, as it was shown previously. In this case it will be a low pass filter whose sample rate will be the same as the system and the decimation will be 100 and because of that, the sample rate at this point will be 200 KHz. It is useful to remember that the interpolation multiplies the rate while the decimation divides it.

![Image 61](image61.png)

*Image 61 - A filter was added to the design*

Based on the properties mentioned, the values of this filter could be:
There will also be a rational resampler. It will adapt the notion rate of a stream by interpolating and decimating the stream. This is necessary because the HackRF rate is a rational multiple of the audio sink rate.

The sample rate of the audio sink is typically 48000 Hz, which is used by professional digital video equipment. If 200 KHz is multiplied by 12 and divided by 5, it will be 480 KHz, which is a multiple of 48 KHz.
The connection of the blocks should look like this example:

![Image 64 - Connections of the rational resampler](image)

The next block will be the demodulator. Since the purpose is to receive wide band FM, the block should be a WBFM Receiver that will have a quadrature rate of 480 KHz according to the rate that was calculated previously, and the decimation will be 10. This value will divide the sample rate by 10 and it will become 48 KHz, that was the desired sample rate.

![Image](image)

Questions

a) What would happen if the resampling does match the sampling rate of the signal with the quadrature rate of the demodulator?

This system will produce an “audible” output. It will have a sample rate enough to be saved as a file or to be reproduced by an audio card. The first option is done using a wave file sink and the second option is done using an audio sink. In both cases, the audio card will convert the signals from digital to analog:
The path of the wave file sink should be an absolute path, for example:

/home/student/my_radio.wav
Optionally, it would help to add a sink to visualize the frequency received by the HackRF. It would be as easy as adding a WX GUI FFT Sink at the output of the osmocom source:

![Image 66 - A FFT sink to visualize the signal received by the SDR peripheral](image)

The high central peak corresponds to the direct current (DC) component:
Adrian Crespo

Digital transmission and reception techniques with Software Defined Radio

It could be removed using a DC Blocker block, but If GNU Radio is running on a Live USB the performance seems to be quite affected by this block.

Questions

a) Implement a FM receiver and record your favourite radio station.
b) Try to remove the DC component to visualize the frequencies properly.
c) Propose a hypothetic diagram for FM transmission. Which blocks would be needed for such design?
3.8 ASK Transmission and reception

3.8.1 Introduction

This exercise will show how to work with digital transmissions. The design will use a ASK modulation. ASK is a form of amplitude modulation that represents digital data as variations in the amplitude of a carrier wave.

This example will use a Varicode encoder and decoder. It is an optimal prefix code that supports all ASCII characters but those characters which are the most used in English has shorter codes, makes the algorithm quite useful to shrink and transmit text. The Varicode encoder and decoder used in this example was written by Clayton Smith and released under the GNU General Public License version 3 (Smith, 2014).

The transmitter will use the Varicode encoder to encode a text, extracted from the book Don Kichot z La Manchy, and it will transmit the text using a ASK modulation. The receiver will receive and save it into a text file.

3.8.2 Installing the Varicode encoder-decoder

Firstly, it is needed to install the Varicode encoder-decoder. As it was shown in a previous laboratory, it needs to be compiled and installed. The command “cmake” will prepare the compilation for the Varicode block:

```
student@ubuntu:~/gr-ham/build

student@ubuntu:~/gr-ham/build

Image 67 - Compilation of the Varicode block
```
After that, the commands “make” and “make install” will compile and install the block into GNU Radio Companion. The installation should be executed as an administrator:

![Image 68 - Installation of the Varicode block](image)

After this process finishes, the block will be available under the Ham category:

![Image 69 - The Varicode blocks will be under the Ham category](image)
3.8.3 Designing the transmitter

The transmitter will use some functions that will require the “numpy” module. In order to import that module, it is required to use an “Import block” block.

There will be three variables: the sample rate, the center frequency and the gain. The gain will be a slider and this time the center frequency will be a fixed frequency that will not be modified during runtime. That will make harder to find a difference in the center frequency between the transmitter and the receiver:
It will be used a new block, “File Source” that will let the system read a binary file. The path of this file must be an absolute path, for example: /home/student/kichot.txt, where kichot.txt will be the fragment extracted from the book “Don Kichot z La Mancha”.

/home/student/kichot.txt,

Using this file, a Varicode encoder block will convert a set of ASCII characters into a digital stream of bytes. But it is still needed another block, named Chunk to Symbols, to obtain a valid signal to be transmitted in ASK modulation.

Questions

a) Find the source code of the Varicode encoder and describe its functionality.
The idea behind this block, Chunk to Symbols, is based on the following assumption. There will be a digital signal after the sound card has processed the I/Q components of the signal processed by the SDR peripheral, through sampling and quantization. However, this digital signal may have a number of levels that does not match ASK:

Therefore, this block chunk to symbols match those levels into a signal that can be used in the ASK modulation. In this case the values for the symbol table that are valid are 0 and 1, since it is ASK.
b) Place a WX GUI Constellation Plot after the Chunk to Symbols block. What does show the plot? What was expected to show?

After that, the sequence will use a special filter. It uses a technique named Frequency Response Masking (FRM) to design a FIR filter. Despite the main advantage of an IIR filter is that the order for the filter is smaller it has some disadvantages: they can become unstable and the coefficient quantification can be more complicated. The filters designed using the Frequency Response Masking technique are named Interpolated FIR filters (IFIR).

The purpose of this filter is to limit the bandwidth of the generated signal. If it changes immediately from off to on (or vice versa) the signal would occupy a large bandwidth. The filter smooths out that transition and therefore limited the bandwidth of the signals.

The Interpolated FIR filter of this exercise, it will convolve an 80-taps Gaussian filter with a square wave. The Gaussian filter will have:

- Gain of 1.
- Symbol rate of 20.
- Bandwidth to bitrate ratio of 1.0.
- 80 taps.

The function to implement this filter would be:
These taps will be multiplied by a constant and there will be a resampler to interpolate the sampling rate by 10:

```
numpy.convolve(numpy.array(filter.firdes.gaussian(1,20,1.0, 4*20)),numpy.array((1,) * 20))
```

Multiplying by 0.9 ensures that the signal remains less than 1 even in the presence of the filter. The SDR peripherals expect the signal to remain between -1 and 1, or else the output will be clipped.

Eventually, the signal will be sent to the SDR peripheral, which will use the sampling rate and the center frequency specified by the variables that were created at the beginning of the design.
Eventually, the whole design should look like this one:

3.8.4 Designing the receiver

Initially, there should be 6 variables: the sampling rate, the center frequency, an offset, a gain, a value for decimation and a value for the correlation. It will be possible to change graphically the gain and the correlation during run-time using a slider:
The center frequency will be close to the frequency of the transmitter. It will be 441 MHz with an offset of 50 KHz and it will have a bandwidth of 2 MHz to ensure that the signal is received by the system. Because of that, it will need to be filtered afterwards.

![Image 81 - Properties of the Osmocom source block](image1.png)

The input signal needs to be multiplied by a cosine acting as a local oscillator, as if it were a super-heterodyne receiver. The intermodulation frequency amplifier of the following diagram would act as the filter and amplifier that will be used in the design.

![Image 82 - Superheterodyne receiver](image2.png)
Adrian Crespo  
*Digital transmission and reception techniques with Software Defined Radio*

The frequency of the local oscillator is set so the desired reception radio frequency \(f_{RF}\) mixes to \(f_{IF}\). The mixer may inadvertently produce additional frequencies and because of that there should be a filter to remove all but the desired signal.

![Image 83 - Multiplication between the signal source and the Osmocom Source](image)

The signal is filtered with a low-pass filter, as mentioned previously, which will have the following properties:

![Image 84 - Properties of the filter](image)
The following blocks are quite easy to understand: there will be one block to obtain the magnitude for the complex signal and there will be another block that must remove the DC component.

After that, there should be a clock data recovery (CDR). The purpose is to recover samples from the received signal at the same frequency and phase as the transmitter since there is not an accompanying clock signal.

By default, GNU Radio includes a Mueller and Müller clock recovery block that will recover the frequency and the phase and decimate the stream. As Tomaž Šolc clarifies (Šolc, 2015), the arguments of this block are:

- Omega is symbol period in samples per symbol.
- The initial value of Mu, which is the detected phase shift in samples between the receiver and transmitter, is not relevant in practice.
- The relative limit parameter determines how much could change omega and because of that a small value would be desired in this exercise.
For this exercise, the values of the Mueller and Müller clock recovery block will be:

![Image 86 - Properties of the clock recovery block](image)

Eventually, the stream will be processed into bytes and the Varicode decoder will recover the ASCII characters that were originally sent:

![Image 87 - Using the Varicode decoder](image)

Since there will be a digital signal at the output of the Varicode decoder, the stream of ASCII characters, be saved into a file. or they can be sent to the standard output:

![Image 88 - Either saving or displaying the text](image)
The whole design should look similar to this one:

![Diagram](Image_89) - The whole design of the receiver

Questions

b) Try to send an image instead of a text file. Explain why the Varicode encoder and decoder is not needed in that case.
Adrian Crespo

Digital transmission and reception techniques with Software Defined Radio
4 Conclusions

Given the sample of digital transmission and reception exercises that can be found in this project, a student could start to study the basics of signals and systems and digital signal processing. The first exercise is a short introduction to Python that will help to those students who are new into this language. It provides the basic notions to be able to read a code and to do some modifications to an existing code.

The second and third exercises try to solve the same problem but with two different perspectives: the first approach tries to introduce the graphical interface to the student and, therefore, some nomenclature that may be easier to associate and to remember given a graphical object. The second approach tries to secure these concepts and associate them to functions and code written in Python. Both exercises complement each other through repeating the same elements and the supported by the same theory.

The fourth exercise is the first time when a student could be able to apply all the previous concepts to design a receiver that will interact with a real environment. The introduction of the Software Defined Radio peripherals and the design of a superheterodyne receiver is the first attempt to associate a project with concepts of modulation and coding.

The last exercise is the most exhaustive, including the design of a transmitter and a receiver. Some concepts of codification, limitation of frequencies and design of filter are used and therefore, this exercise may be the most challenging for a student.

Therefore, given these exercises the student may be able to:

- To understand some transmission and reception techniques.
- To understand some basic concepts of signals and systems, such as the sample ratio, frequency allocation, noise filtering, window functions, filter design, clock recovery, codification, etcetera.
- To apply a modern high-level programming language to transmission and reception systems.
- To test and measure the effect of different parameters and characteristics to real cases.
- To be familiar with some basic designs and circuits, such as the superheterodyne receiver or some basic demodulation processes.
- To conclude, all the exercises include some questions that may encourage the students to research and go deeper in the subject, and this is the final purpose of this project.
5 References


Retrieved from Library of Congress:
http://digital.library.unt.edu/ark:/67531/metacrs7941/m1/1/high_res_d/RL33161_2005Nov17.pdf


http://groups.winnforum.org/d/do/1725


https://www.tablix.org/~avian/blog/archives/2015/03/notes_on_m_m_clock_recovery/

