PhysicsOpenLab
physicsopenlab.org Printed on October 31, 2022

GNURadio Software for the 21 cm Neutral-Hydrogen Line

https://physicsopenlab.org/2020/07/26/gnuradio-software-for-the-21-cm-neutral-hydrogen-line/

July 26, 2020

Abstract : *After the previous posts on the horn antenna (Horn Antenna for the 21cm Neutral-Hydrogen Line) and on the receiver (Low-Noise SDR-Based Receiver for the 21cm Neutral-Hydrogen Line) in this post we briefly describe the* ***Linux GNURadio*** *application and its use for receiving the 21 cm emission of neutral hydrogen.*

Introduction

GNURadio is a free and open source software development toolkit that provides signal processing blocks for implementing radio software projects. It can be used with external RF hardware to create software defined radio (SDR) or even without hardware in a simulation environment. It is widely used in hobby, academic and commercial environments to support both wireless communications research and real world radio systems.

GNURadio is licensed under the GNU General Public License (GPL) version 3 or later. All code is copyright of the Free Software Foundation.

GNURadio uses a combination of Python and C++, where Python handles the high level interface and C++ is used to implement drivers and low level interfaces to the hardware. This combination allows for a system that is easy to use, but still meets the performance required for handling large amounts of data.

GNURadio also has a rapid development tool called **GNU Radio companion (GRC)**. GRC is a simple to use graphical system for designing and building radio components in software.

GNU Radio Companion provides common functions, such as signal sources, signal processing and signal sinks, as blocks that can be picked and placed on the screen. Once placed, the blocks can be wired up, much like in LabView, and the fow of data can be controlled in this fashion. GNU Radio Companion also includes blocks that allow for building a GUI interface, which can be used to display data and control the software defined radio.

If a block does not exist, it can be created. Because GNURadio uses Python, users can use this powerful and fexible language to build new blocks that can be imported into GRC.

GNURadio is a very complete system, which requires a learning period to fully appreciate its complexity and wealth of functionality. We will limit ourselves to the aspects that most directly affect our project. For descriptions, explanations, examples on GNURadio we have collected the following links.

References on GNURadio

Digital Signal Processing in Radio Astronomy : **DSPiRA**

GNURadio Wiki : **wiki.gnuradio.org**

GNURadio Tutorials : **Tutorials**

GNURadio Installation

The GNU Radio application is easily installed on **Linux**. It can be used, with some difficulties, also on other operating systems but the use of Linux is strongly recommended. We have installed it on a **Ubuntu 20.04 Linux distribution**.

We operate from the terminal and use the **apt install** command. First we install some dependencies:

```
sudo apt install git
sudo apt install cmake
sudo apt install python-apt liborc-0.4-dev
```

Then we proceed to install the **gnuradio** application, the **OsmoSDR** drivers for the SDR hardware and for **Airspy** :

```
sudo apt install gnuradio gr-osmosdr limesuite airspy python3-h5py python3-ephem
```

At this point the installation of the last versione of GNURadio (now 3.8) should be completed. By connecting Airspy to the USB socket and typing the command

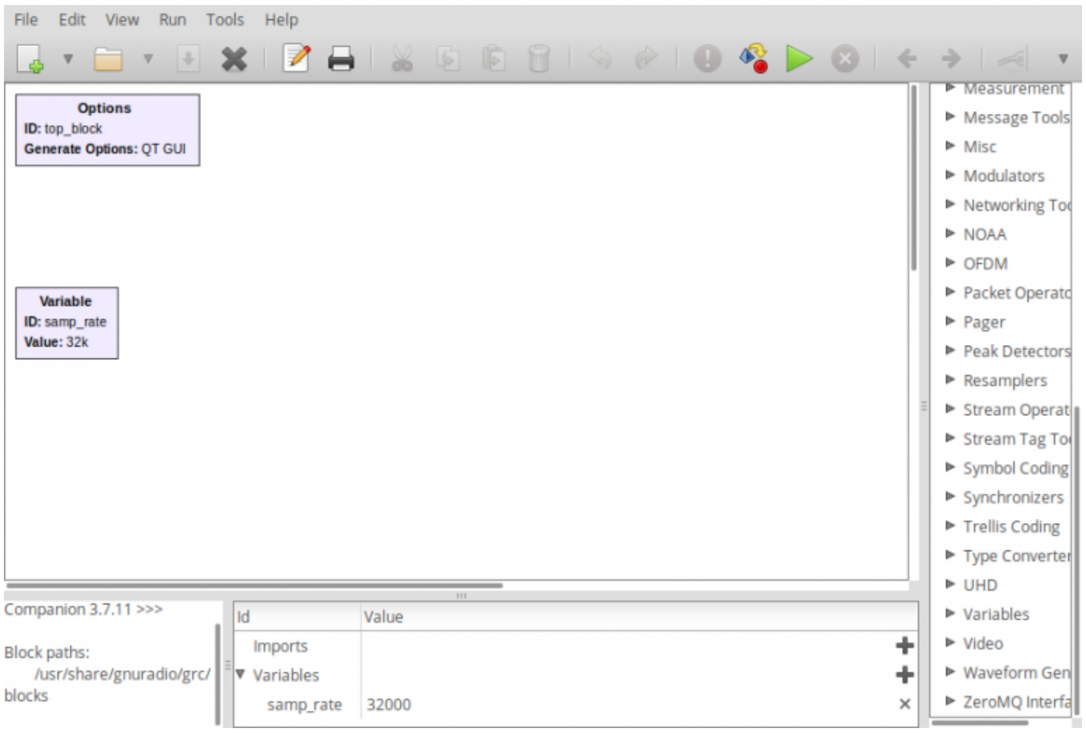
```
airspy_info
```

recognition by the PC should be verified and the main card data obtained.

To start the GNURadio application just type the following command from the terminal:

```
gnuradio-companion
```

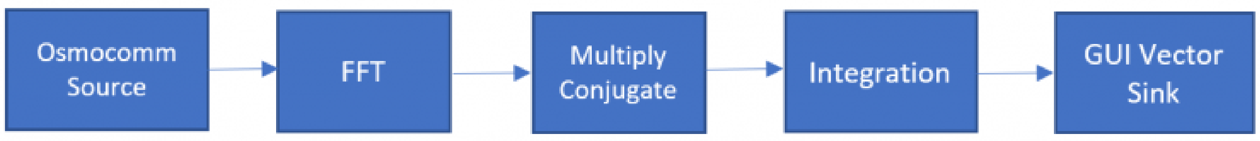
This opens GNU Radio Companion (GRC) :



```
Companion 3.7.11 >>>

Block paths:
     /usr/share/gnuradio/grc/
blocks
```

A Basic Spectrometer

The GNURadio application we need is basically a **radiospectrometer**. The signal picked up by the antenna and subsequently amplified and filtered by the RF components is sent to the SDR receiver which acquires it in digital format. At this point the signal must be processed in order to calculate its power spectrum and highlight the signal peak corresponding to the emission at 21cm. The spectrometer, in its simplest form can be described by the following block diagram :



- **Osmocomm Source** : is the driver for the Airspy R2 SDR which is the source of the signal.
- **FFT** : Fourier transform FFT, determines the frequencies present in the input signal.
- **Multiply Conjugate** : calculates the square modulus of the FFT to determine the signal strength at each frequency
- **Integration** : carries out the integration of the signal (sum) over a time interval (0.1s – 10s) to minimize noise
- **GUI Vector Sink** : displays the spectrum of the signal on the screen

An Advanced Spectrometer : spectrometer_w_cal.grc

For the processing of the RF signal acquired by the airspy R2 SDR receiver we used the GNURadio program spectrometer_w_cal.grc which is part of a larger project of the **DSPiRA** working group. This is a truly remarkable project of the research group in Radio Astronomy of the University of West Virginia which aims to make available to enthusiasts, teachers and students a set of documentation and tools for the practice of radio astronomy research.

Thanks to the authors of the software and in particular to Kevin Bandura, Glen Langston and Pranav Sanghavi.

Main Repository : **https://github.com/WVURAIL/gr-radio_astro**

Branch for Ubuntu 20.04 / GNURadio 3.8 : **https://github.com/WVURAIL/gr-radio_astro/tree/gr38**

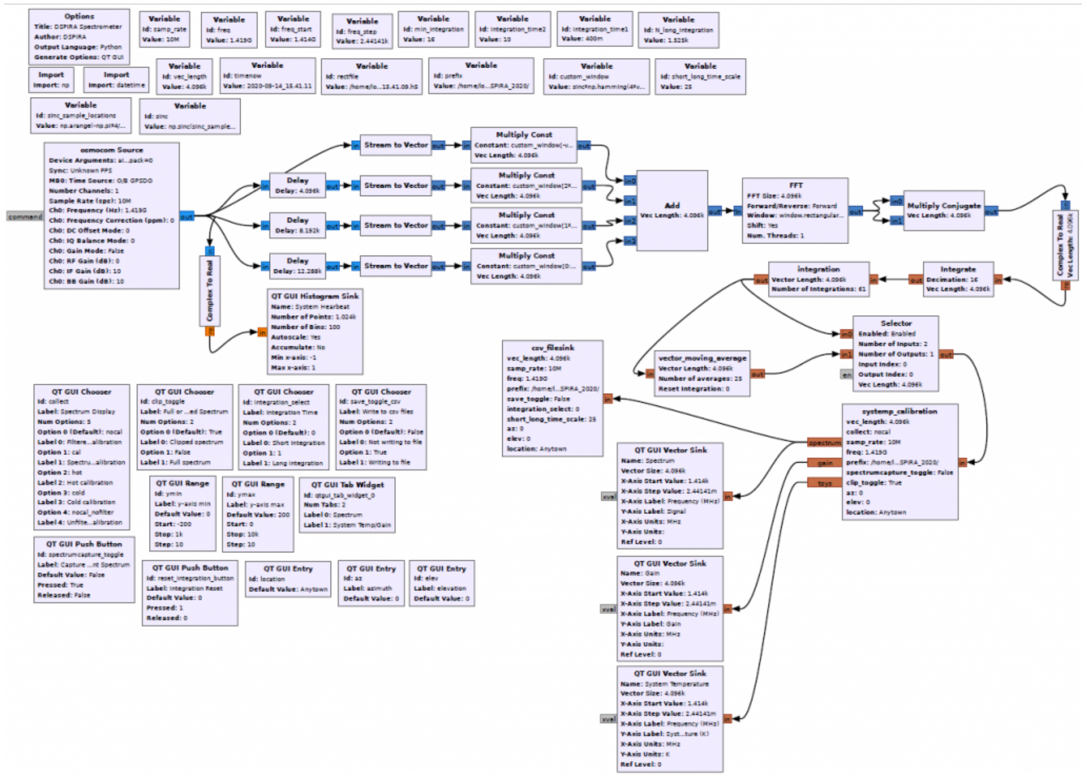
Wiki for installing : **https://github.com/WVURAIL/gr-radio_astro/wiki/Installing-gr-radio_astro**

The software allows the acquisition of the signal and the determination of its power spectrum. In particular, there are the calibration function with the acquisition of the “Hot” and “Cold” spectrum. The spectrum is displayed on the screen and can be saved to a file in csv format. The gain and equivalent system temperature are also displayed on the screen. Furthermore, the integration time of the signal can be chosen between two values, by default set at 0.1s and 10s.

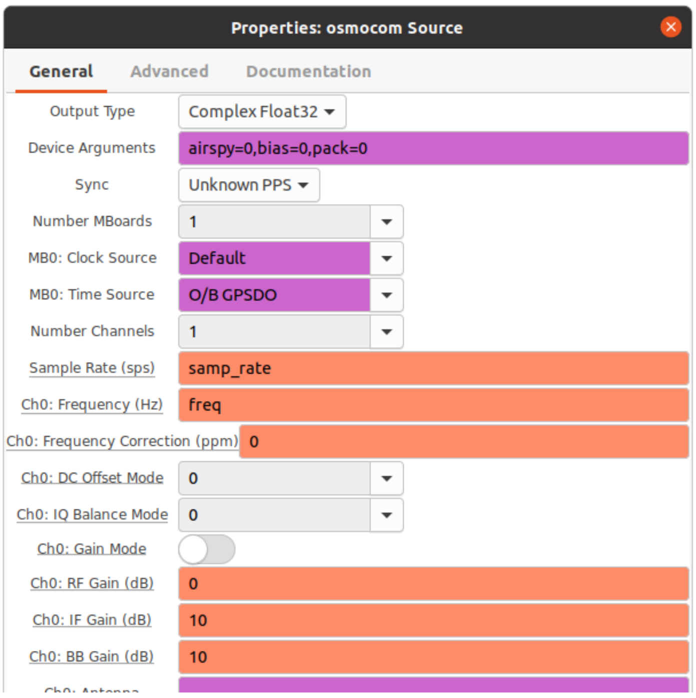
The diagram below shows the entire block diagram of the program as it is displayed by GNURadio. GNURadio also generates a python executable. The program can be run by launching the corresponding python file or by opening the .grc source in GNURadio and running it within the GNURadio environment. We prefer to go for GNURadio in order to have more control over the process.

There are some variables to be configured according to the environment and others that can be changed to modify the spectrometer :

- **rectfile** : path for file writing **(to be configured)**
- **prefix** : path for csv file writing **(to be configured)**
- **integration_time1** : short integration time (default 0,4s)
- **integration_time2** : long integration time (default 10s)
- **samp_rate** : sample rate (airspy has two options : 10M and 2.5M)
- **freq** : central frequency (default 1.419G)
- **vec_length** : samples vector length (default 4096)


```
Options
Title: DSPIRA Spectrometer
Author: DSPIRA
Output Language: Python
Generate Options: QT GUI

Variable
Id: samp_rate
Value: 10M

Variable
Id: freq
Value: 1.419G

Variable
Id: freq_start
Value: 1.414G

Variable
Id: freq_step
Value: 2.44141k

Variable
Id: min_integration
Value: 16

Variable
Id: integration_time2
Value: 10

Variable
Id: integration_time1
Value: 400m

Variable
Id: N_long_integration
Value: 1.525k

Import
Import: np

Import
Import: datetime

Variable
Id: vec_length
Value: 4.096k

Variable
Id: timenow
Value: 2020-09-14_16.41.11

Variable
Id: rectfile
Value: /home/lo...16.41.09.h5

Variable
Id: prefix
Value: /home/lo...SPIRA_2020/

Variable
Id: custom_window
Value: sinc*np.hamming(4*v...

Variable
Id: short_long_time_scale
Value: 25

Variable
Id: sinc_sample_locations
Value: np.arange(-np.pi*4/...

Variable
Id: sinc
Value: np.sinc(sinc_sample...

osmocom Source
Device Arguments: ai...pack=0
Sync: Unknown PPS
MB0: Time Source: O/B GPSDO
Number Channels: 1
Sample Rate (sps): 10M
Ch0: Frequency (Hz): 1.419G
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 10
Ch0: BB Gain (dB): 10

Delay
Delay: 4.096k

Delay
Delay: 8.192k

Delay
Delay: 12.288k

Stream to Vector

Multiply Const
Constant: custom_window[-v...
Vec Length: 4.096k

Multiply Const
Constant: custom_window[2*...
Vec Length: 4.096k

Multiply Const
Constant: custom_window[1*...
Vec Length: 4.096k

Multiply Const
Constant: custom_window[0:...
Vec Length: 4.096k

Add
Vec Length: 4.096k

FFT
FFT Size: 4.096k
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

Multiply Conjugate
Vec Length: 4.096k

Complex To Real
Vec Length: 4.096k

Integrate
Decimation: 16
Vec Length: 4.096k

integration
Vector Length: 4.096k
Number of Integrations: 61

Selector
Enabled: Enabled
Number of Inputs: 2
Number of Outputs: 1
Input Index: 0
Output Index: 0
Vec Length: 4.096k

vector_moving_average
Vector Length: 4.096k
Number of averages: 25
Reset Integration: 0

csv_filesink
vec_length: 4.096k
samp_rate: 10M
freq: 1.419G
prefix: /home/l...SPIRA_2020/
save_toggle: False
integration_select: 0
short_long_time_scale: 25
az: 0
elev: 0
location: Anytown

systemp_calibration
vec_length: 4.096k
collect: nocal
samp_rate: 10M
freq: 1.419G
prefix: /home/l...SPIRA_2020/
spectrumcapture_toggle: False
clip_toggle: True
az: 0
elev: 0
location: Anytown

Complex To Real

QT GUI Histogram Sink
Name: System Hearbeat
Number of Points: 1.024k
Number of Bins: 100
Autoscale: Yes
Accumulate: No
Min x-axis: -1
Max x-axis: 1

QT GUI Vector Sink
Name: Spectrum
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Signal
X-Axis Units: MHz
Y-Axis Units:
Ref Level: 0

QT GUI Vector Sink
Name: Gain
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Gain
X-Axis Units: MHz
Y-Axis Units:
Ref Level: 0

QT GUI Vector Sink
Name: System Temperature
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Syst...ture (K)
X-Axis Units: MHz
Y-Axis Units: K
Ref Level: 0

QT GUI Chooser
Id: collect
Label: Spectrum Display
Num Options: 5
Option 0 (Default): nocal
Label 0: Filtere...alibration
Option 1: cal
Label 1: Spectru...alibration
Option 2: hot
Label 2: Hot calibration
Option 3: cold
Label 3: Cold calibration
Option 4: nocal_nofilter
Label 4: Unfilte...alibration

QT GUI Chooser
Id: clip_toggle
Label: Full or ...ed Spectrum
Num Options: 2
Option 0 (Default): True
Label 0: Clipped spectrum
Option 1: False
Label 1: Full spectrum

QT GUI Chooser
Id: integration_select
Label: Integration Time
Num Options: 2
Option 0 (Default): 0
Label 0: Short Integration
Option 1: 1
Label 1: Long Integration

QT GUI Chooser
Id: save_toggle_csv
Label: Write to csv files
Num Options: 2
Option 0 (Default): False
Label 0: Not writing to file
Option 1: True
Label 1: Writing to file

QT GUI Range
Id: ymin
Label: y-axis min
Default Value: 0
Start: -200
Stop: 1k
Step: 10

QT GUI Range
Id: ymax
Label: y-axis max
Default Value: 200
Start: 0
Stop: 10k
Step: 10

QT GUI Tab Widget
Id: qtgui_tab_widget_0
Num Tabs: 2
Label 0: Spectrum
Label 1: System Temp/Gain

QT GUI Push Button
Id: spectrumcapture_toggle
Label: Capture ...nt Spectrum
Default Value: False
Pressed: True
Released: False

QT GUI Push Button
Id: reset_integration_button
Label: Integration Reset
Default Value: 0
Pressed: 1
Released: 0

QT GUI Entry
Id: location
Default Value: Anytown

QT GUI Entry
Id: az
Label: azimuth
Default Value: 0

QT GUI Entry
Id: elev
Label: elevation
Default Value: 0
```

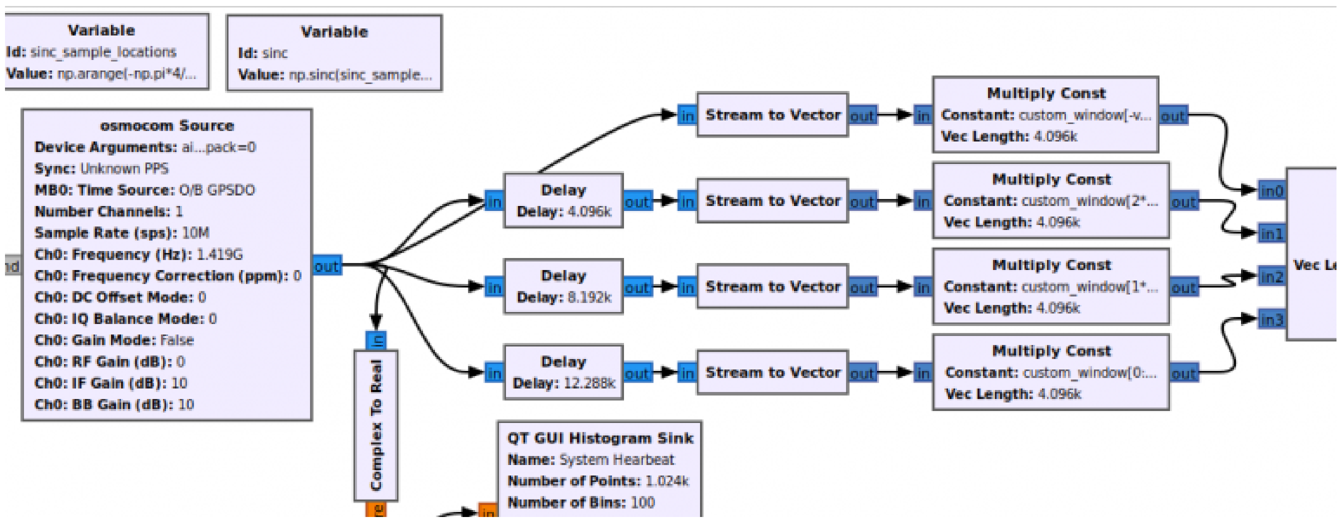
For details on the software, please refer to the **DSPiRA** website and the documentation found on the repository, we will limit ourselves here to some notes on some general aspects. The **osmocomm Source** block is the interface to the Airspy R2 SDR hardware. In our case we don't use the bias-T option so in the argument string we put "*airspy=0,bias=0,pack=0*", the Sample Rate and Frequency parameters take the value from the program variables. The RF gain is set to 0, while the IF and BB gains are set to 10dB. We have found that this gain configuration with our RF hardware allows for good system utilization. With different amplification chains, these parameters can, of course, be varied in order to optimize performance.


Properties: osmocom Source

General | Advanced | Documentation

Output Type	Complex Float32
Device Arguments	airspy=0,bias=0,pack=0
Sync	Unknown PPS
Number MBoards	1
MB0: Clock Source	Default
MB0: Time Source	O/B GPSDO
Number Channels	1
Sample Rate (sps)	samp_rate
Ch0: Frequency (Hz)	freq
Ch0: Frequency Correction (ppm)	0
Ch0: DC Offset Mode	0
Ch0: IQ Balance Mode	0
Ch0: Gain Mode	
Ch0: RF Gain (dB)	0
Ch0: IF Gain (dB)	10
Ch0: BB Gain (dB)	10

The first step in the signal processing chain is a “polyphase filter”. The **polyphase filter (PFB)** is a DSP technique that aims to mitigate the drawbacks of simple DFT. The PFB produces a flat response throughout the channel and provides excellent suppression of out-of-band signals. This filter is made by splitting the stream of samples into four vectors, each of these is multiplied by a suitable constant and then added to the others to produce a single vector on which our FFT will then be applied. For details on this technique, refer to the DSPiRA website.


Variable
Id: sinc_sample_locations
Value: np.arange(-np.pi*4/...

Variable
Id: sinc
Value: np.sinc(sinc_sample...

osmocom Source
Device Arguments: ai...pack=0
Sync: Unknown PPS
MB0: Time Source: O/B GPSDO
Number Channels: 1
Sample Rate (sps): 10M
Ch0: Frequency (Hz): 1.419G
Ch0: Frequency Correction (ppm): 0
Ch0: DC Offset Mode: 0
Ch0: IQ Balance Mode: 0
Ch0: Gain Mode: False
Ch0: RF Gain (dB): 0
Ch0: IF Gain (dB): 10
Ch0: BB Gain (dB): 10

Delay
Delay: 4.096k

Delay
Delay: 8.192k

Delay
Delay: 12.288k

Stream to Vector

Multiply Const
Constant: custom_window[-v...
Vec Length: 4.096k

Multiply Const
Constant: custom_window[2*...
Vec Length: 4.096k

Multiply Const
Constant: custom_window[1*...
Vec Length: 4.096k

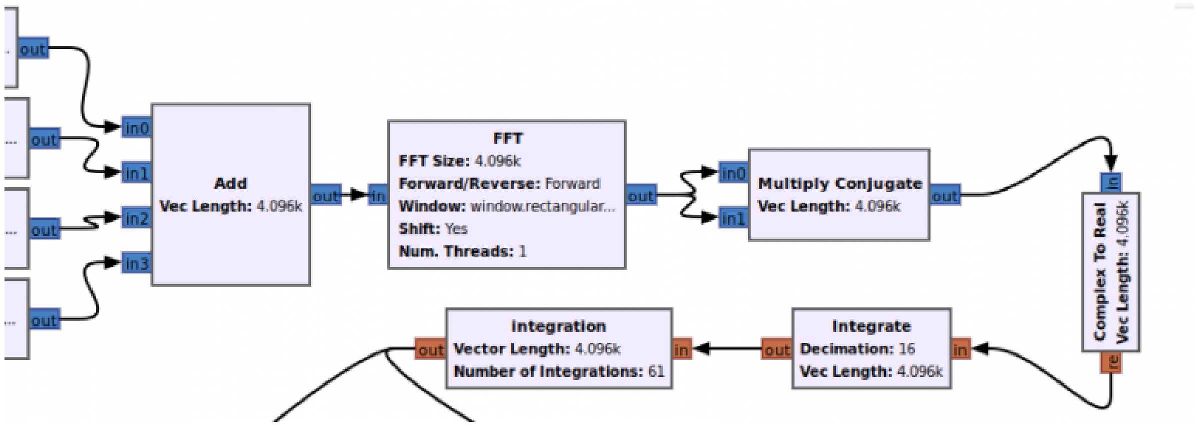
Multiply Const
Constant: custom_window[0:...
Vec Length: 4.096k

Complex To Real

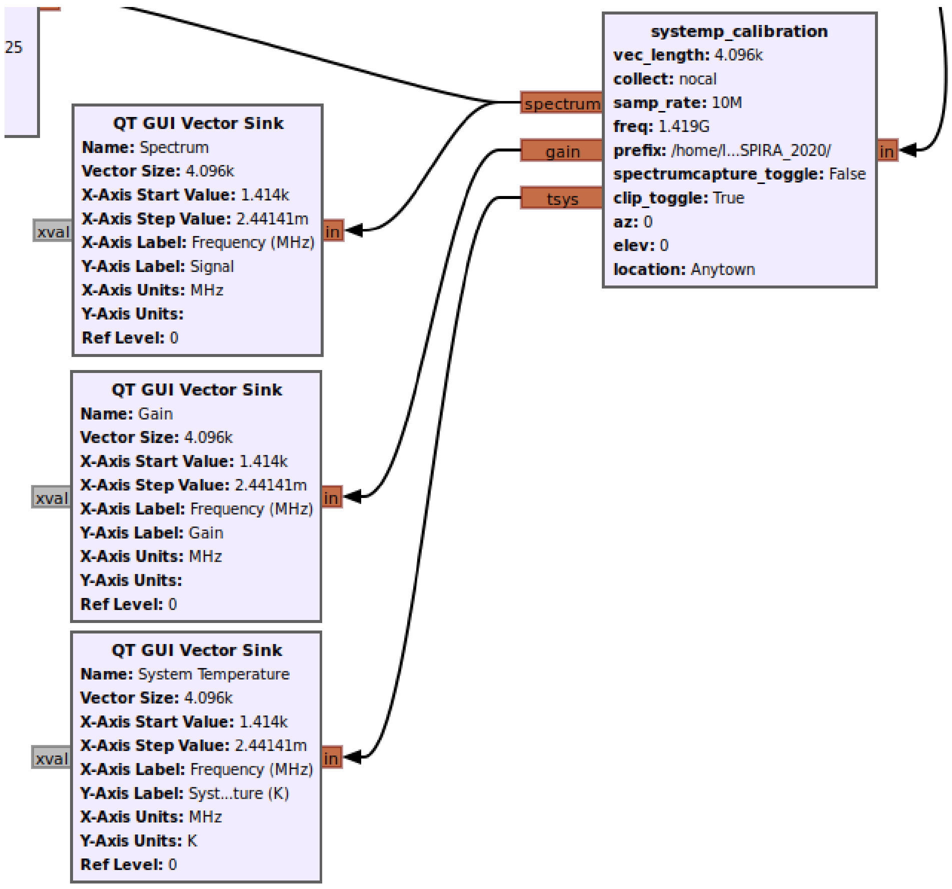
QT GUI Histogram Sink
Name: System Hearbeat
Number of Points: 1.024k
Number of Bins: 100

Downstream of the “polyphase filter” the vector with the data is sent to the main blocks which execute respectively :

- **FFT** : Fourier transform FFT, determines the frequencies present in the input signal.
- **Multiply Conjugate** : calculates the square modulus of the FFT to determine the signal strength at each frequency
- **Integration** : carries out the integration of the signal (sum) over a time interval (0.1s – 10s) to minimize noise



After the time integration operations, the vector with the spectrum values is sent to the custom **systemp_calibration** block which performs the calibration operations. For details on the calibration of the radio telescope, refer to the next paragraph. The outputs of this block are the three vectors of the **spectrum**, **gain** and **system temperature** which are sent to three respective **QT GUI Vector Sinks** for on-screen display. The spectrum vector is also sent to a block that writes to a file in csv format.

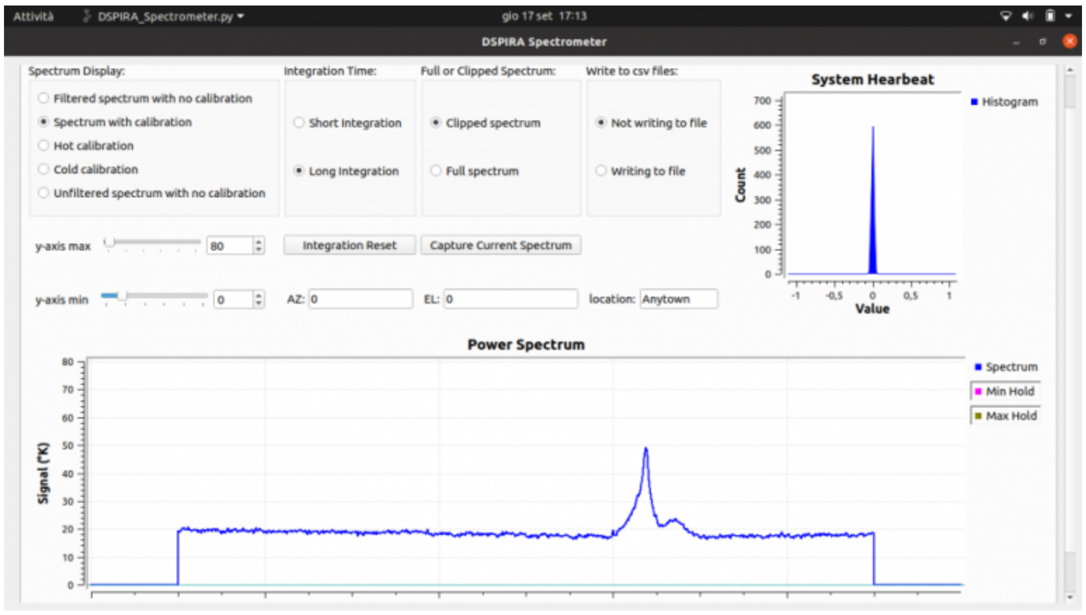

systemp_calibration
vec_length: 4.096k
collect: nocal
samp_rate: 10M
freq: 1.419G
prefix: /home/l...SPIRA_2020/
spectrumcapture_toggle: False
clip_toggle: True
az: 0
elev: 0
location: Anytown

QT GUI Vector Sink
Name: Spectrum
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Signal
X-Axis Units: MHz
Y-Axis Units:
Ref Level: 0

QT GUI Vector Sink
Name: Gain
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Gain
X-Axis Units: MHz
Y-Axis Units:
Ref Level: 0

QT GUI Vector Sink
Name: System Temperature
Vector Size: 4.096k
X-Axis Start Value: 1.414k
X-Axis Step Value: 2.44141m
X-Axis Label: Frequency (MHz)
Y-Axis Label: Syst...ture (K)
X-Axis Units: MHz
Y-Axis Units: K
Ref Level: 0

The following image shows the screenshot of the running program in which you can see, in the window with the spectrum graph, our peak corresponding to the emission at 21 cm !



Calibration

The minimum intensity of radiation that can be collected by a radio telescope antenna depends on the noise associated with the receiving system. In addition to the useful signal, unwanted components of various nature and origin, such as *noise* and *disturbances*, are generally measured. Total noise is divided into two fundamental contributions: *antenna noise* and *equipment noise*. The useful radio astronomy signal is also constituted by noise with a Gaussian distribution, this allows all the contributions (signal + noise) to be summed in power in order to obtain an overall value called *average power of equivalent noise* :

$$P_e = k*T_e*B$$

k = Boltzmann Constant

B = Receiving Band

Which leads to an *equivalent antenna temperature* :

$$Te = P_e / (k*B)$$

To use our radio telescope efficiently and to obtain accurate measurements, however, we need to separate the contributions of the signal from those of the noise: this is done with the **calibration procedure**. If we assume that the behavior of the system is **linear** then it is sufficient to measure on two points, with known temperatures and sufficiently distant from each other. The measurement is usually carried out on a **cold point**, for example by pointing the antenna to the **zenith**, in the absence of radio sources, where the brightness temperature has a value of **10°K**; and pointing the antenna towards a **hot point**, for example the **ground**, for which a brightness temperature of about **300°K** is assumed.

In detail we can write :

$$P_{measured} = G*(T_{object} + T_{system})$$

Where

- $P_{measured}$ = Signal detected with radiotelescope
- G = gain (constant of proportionality between the measured power and the brightness temperature)
- T_{object}= Signal coming from the object to be measured (for example the galaxy)
- T_{system}= "Noise" signal due to other disturbing sources

Moreover the parameters G, T_{object} e T_{system} depends on the frequency. As we explained above, calibration consists in measuring and acquiring the power spectrum for two known temperatures :

$T_{object} = T_{hot} = 300°K$

$T_{object} = T_{cold} = 10°K$

With these data we can calculate the parameters of our telescope G and T_{system} . From relation we can obtain the following relations :

$$G = (P_{hot} - P_{cold})/(T_{hot} - T_{cold})$$

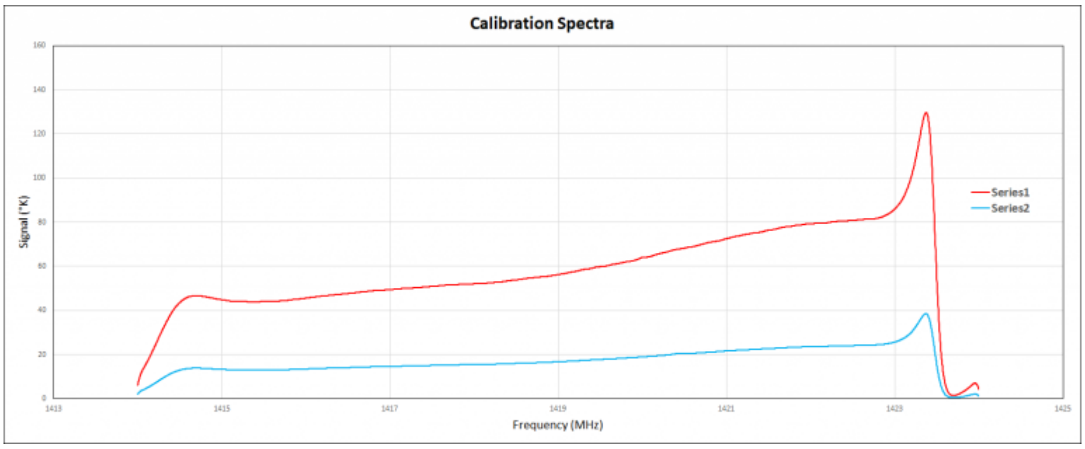
$$T_{system} = (T_{hot} - T_{cold}(P_{hot}/P_{cold}))/((P_{hot}/P_{cold})-1)$$

Knowing these parameters we can easily obtain the data that interests us :

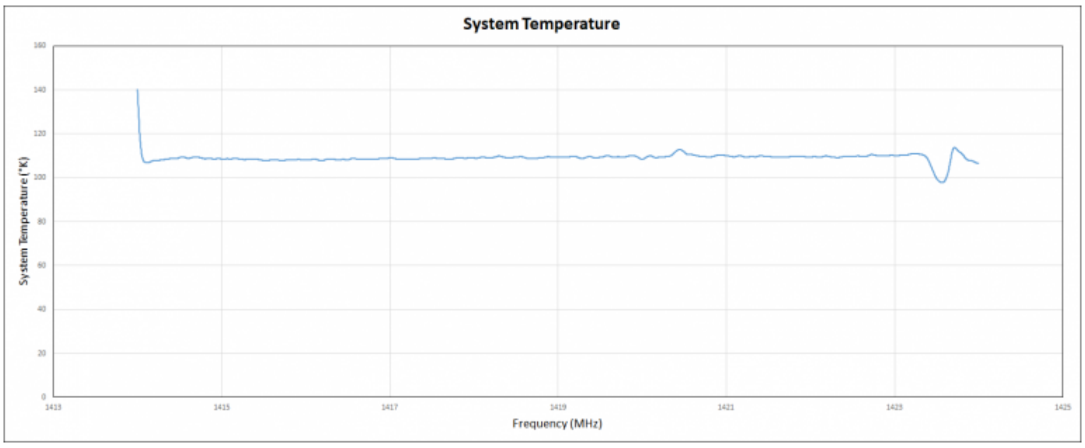
T_{object}

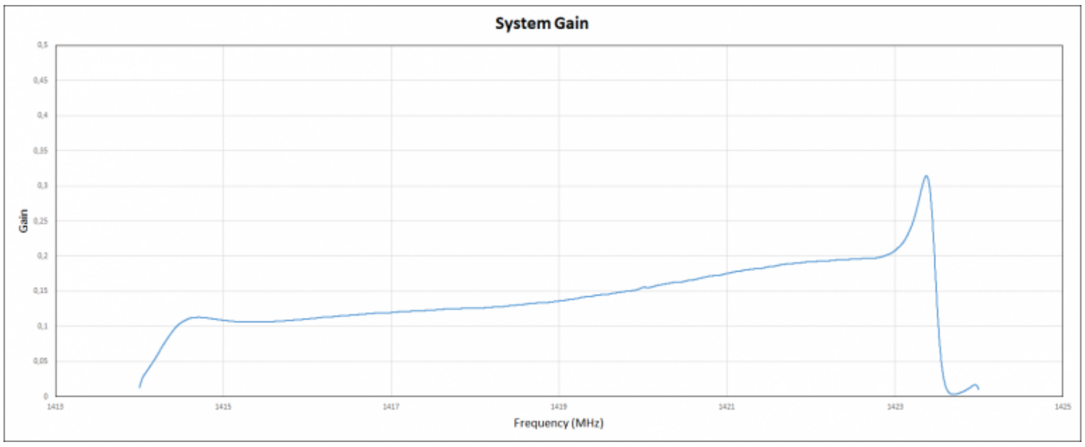
$$T_{object} = P_{measured} / G - T_{system}$$

The chart below shows the calibration spectra : red = hot, blue = cold.

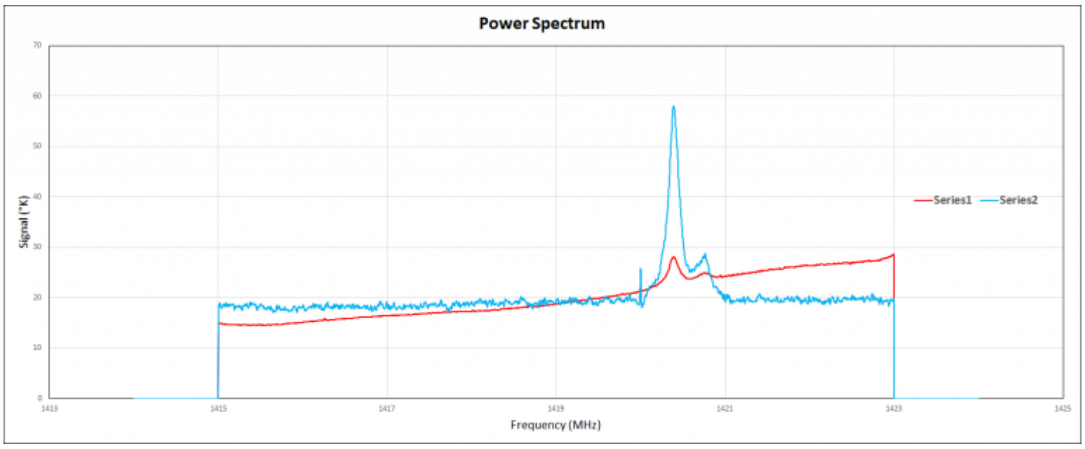


From calibration spectra we obtain the trend of the system temperature T_{system} and gain G. The equivalent system temperature stands at a value of approximately **110°K**. From the analysis of the receiver we have obtained (in the previous post) an equivalent temperature of 58°K, which shows that the antenna temperature is the difference, thus **52°K**.



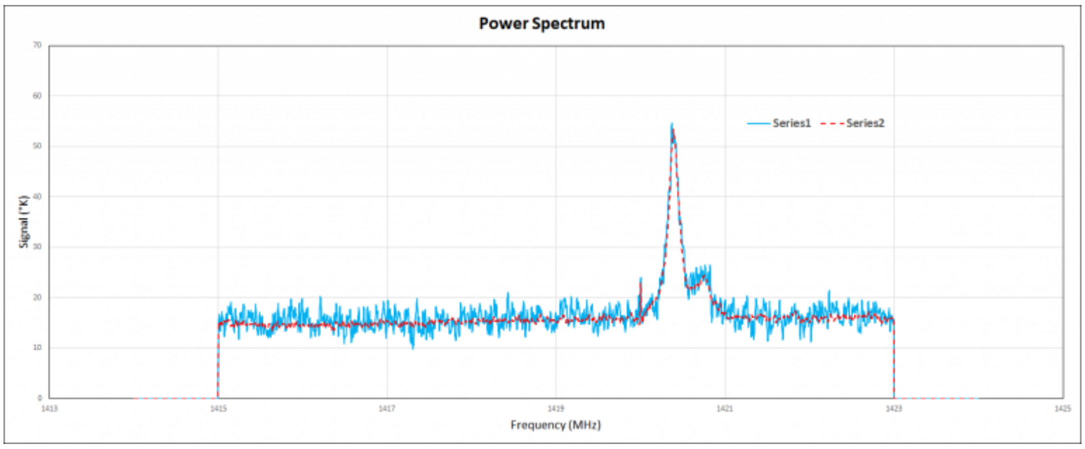


The graphs below show the spectra obtained by pointing the antenna towards an area of the Milky Way. In red the uncalibrated spectrum, in blue the same spectrum, however, calibrated with the previously obtained G and Tsys parameters : as you can see, the calibrated spectrum makes the radio emissions of the galaxy much more evident and accurate.



First Light

After calibration we can finally use our radio telescope to detect the emissions at 21cm of the hydrogen clouds present in our galaxy. The GNURadio spectrometer_w_cal program allows the use of two signal integration time constants: 0.1s and 10s. For a quick view of the signal, for example while we are pointing the antenna, it may be useful to use the fast integration in order to immediately see the signal trend; for the acquisition of the spectrum (and for the calibration) it is advisable to use the long integration in order to have a cleaner signal, as you can see in the graph below where we compare, in blue the integration at 0.1s and in red the signal which is obtained with the integration at 10s.



Conclusions

Now our radio telescope for detecting the emission at 21cm is ready : we will use it for the study of the structure of our galaxy and for the analysis of its rotation speed.

If you liked this post you can share it on the “social” **Facebook**, **Twitter** or **LinkedIn** with the buttons below. This way you can help us! **Thank you !**

Donation

If you like this site and if you want to contribute to the development of the activities you can make a donation, **thank you !**

Donate