Atacama
Large
Millimeter /
submillimeter
Array

Two Station Interferometer

Documentation

Louise Chantal Dauvin Gutiérrez

lcdauvin@uc.cl

February, 2016

Table of Contents

1.- Introduction

At the time, human being has done a precious contribution to the science and technological development, which has, no doubt, given us a better understanding of the world. In that way, Astronomy is one of the disciplines that aims to bring information of the universe and its beginnings. The instrumentation developed has allowed to discover and propose a great number of theories; nevertheless, there is so much more information to obtain from the sky.

For that reason, it is indispensable to develop new instrumentation and instruct human capital in the area. That are the objectives of this project, which is involved in the EDUALMA plan. It is an interferometer for educational purposes, which should be of low cost and demonstrates an experiment of easy understanding for new students.

The interferometer hardware was composed of low cost elements and its software was based on open source library to Python: GNU Radio. A more detailed explanation of the instrument will be exposed in the following report. In the same way, it is showed the theoretical framework of the experiment that was developed, which is the calculation of the Sun angular diameter.

2.- Theoretical framework

An interferometer is an instrument used in astronomy to combine two signals acquired from different telescopes. The importance resides in the fact that both receptors could have a great separation in order to achieve a higher resolution in the sky. The separation is compared to the size of a telescope, but in this case a big size is achieved by avoiding the manufacturing issues of a big dish. As its name says an interferometer is based on the interference wave phenomena that will be explained in the following lines.

2.1.- Interferometry

Interference is related to the phase of two waves of the same wavelength that are added. If both waves are in phase, the amplitude of the sum will be amplified two times; and if these two waves have a gap of half of a wavelength, the amplitude will be zero, as a result of opposite values sum. This is illustrated in figure 2.2.1.

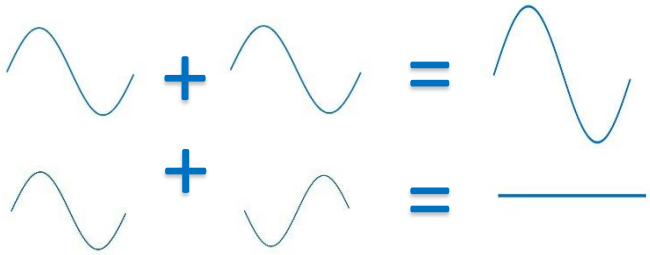


Figure 2.1.1: Constructive interference and destructive interference

An astronomical source produces a flat and coherent wavefront; thus, if the distance from the source is the same or is a multiple of the wavelength in two points, there will be constructive interference in the addition of the two points signals. That is what an interferometer does, two receptors are placed at a given distance and both observe the same source. Depending on the angle of observation, power will increase or decrease and it will form fringes patterns if an angle scan is done. When sun passes over the interferometer, produces the interference effect since it forms different angles.

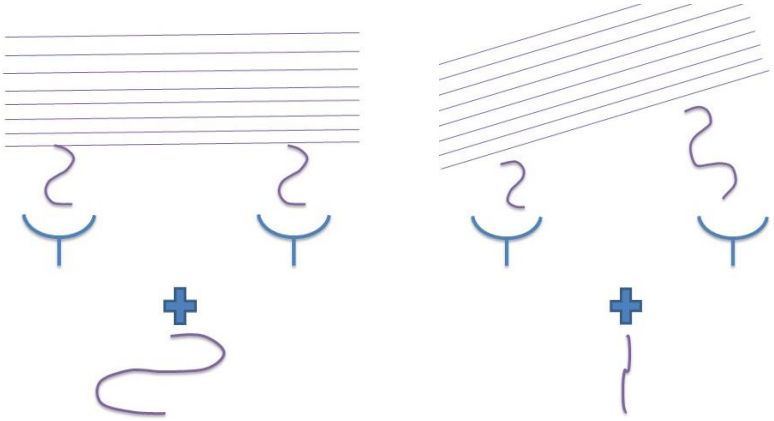


Figure 2.1.2: Constructive interference and destructive interference in an interferometer. When sun passes in different angles above, optical path will be different in each receptor.

2.2.- Sun diameter calculations

Let "B" be the baseline of the interferometer and "α" the angle between receptor pointing and the observed object, as it is showed in figure 2.2.1.

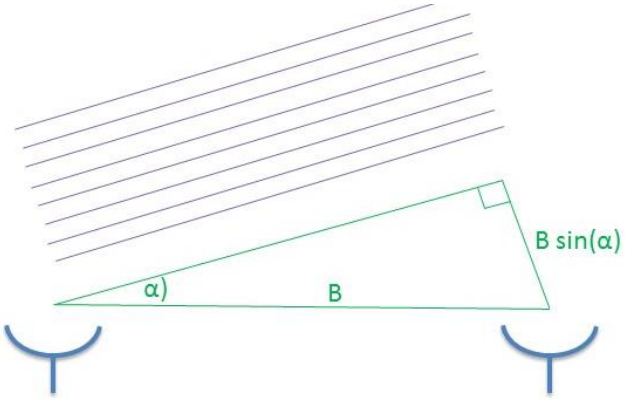


Figure 2.2.1: The flat wavefront arrives in an angle of "α", and produces a spatial delay of "Bsin(α)" between the receptors

The time delay "τ" given when the wavefront arrives to both receptors is related to the spatial delay and the light velocity (since the signals from sky are light):

$$\tau = \frac{B \sin \alpha(\theta_0)}{c}$$

Where $\alpha = \theta - \theta_0$, θ is the receptor pointing and θ_0 the source angle in the sky. Knowig that the star light is an electromagnetic wave, the radiation that it produces can be written through Maxwell equations; for each receptor:

$$\overrightarrow{E_1} = \vec{E}(\theta_0)\, cos(2\,\pi\, \upsilon\, t)$$

$$\overrightarrow{E_2} = \vec{E}(\theta_0)\, cos(2\,\pi\, \upsilon\, (t - \tau))$$

The addition interferometer, precisely sums both signals, hence, its final power will result in:

$$P(t) = \left|\overrightarrow{E_{total}}^2\right| = \left|\left(\overrightarrow{E_1} + \overrightarrow{E_2}\right)^2\right|$$

Thus:

$$P(t) = E^2(\theta_0)\big(1 + cos(2\,\pi\, \upsilon\, \tau)\big)$$

As the Sun is an extended object, it is described as many punctual sources in a given angular diameter. Therefore, the integral of density power distribution for a punctual source will result in the power of the extended source:

$$P(\theta) = \int \varepsilon(\theta_0)\left(1 + cos\left(2\,\pi\,\upsilon\,\tau(\theta_0)\right)\right)\,d\theta_0$$

The fringes disappearance is produced every time the cosine function is -1, hence:

$$cos\left(2\,\pi\,\upsilon\,\tau(\theta_0)\right) = -1$$

Replacing "τ" and assuming a little "α" ($\alpha \approx \sin\alpha$) and a wavelength "$\lambda$", the previous equation is reduced:

$$2\,\pi\,\frac{B\ \alpha}{\lambda} = \ (2n+1)\pi$$

Thus, the period between the minimums is "$\Delta\,\alpha$"

$$2\ \ \frac{B\,\Delta\,\alpha}{\lambda} = \ (2n_1+1) - (2n_2+1)$$

And finally,

$$\Delta\,\alpha = \frac{\lambda}{B}\ [rad]$$

As the measurement in angle of the Sun is done through time, it is necessary to transform that data into radians. The maximum velocity of the sun is given by the earth rotation, i.e. 360° in 24 hours, however, declination will decrement this value as follows:

$$Sun\ velocity = v = \ \frac{360\frac{\pi}{180}\ [rad]}{24 * 3600\ [s]}\ cos(declination)$$

Thus, the period in time units is useful to calculate baseline and it is given by:

$$Period\ = \frac{\lambda}{B\,v}$$

For Sun diameter calculation, it is needed the power equation for an extended source, which is written as:

$$P(\theta) = \ \int \varepsilon(\theta_0) d\theta_0\ \ (1+V)$$

Where the spectrum of the sun is:

$$S = \ \int \varepsilon(\theta_0) d\theta_0$$

and

$$V = \frac{1}{S}\int \varepsilon(\theta_0)\cos\left(2\pi\frac{B}{\lambda}(\theta-\theta_0)\right)d\theta_0$$

$$V = \frac{cos\left(2\pi\frac{B}{\lambda}\theta\right)}{S}\int \varepsilon(\theta_0)e^{-i\,2\pi\frac{B}{\lambda}\theta_0}$$

is the visibility function. Hence, power is written as:

$$P(\theta) = S\left(1 + Vo\ cos\left(2\pi\frac{B}{\lambda}(\theta - \Delta\theta)\right)\right)$$

Thus, the maximum of power will be achieved in a peak value of cosine function:

$$Pmax = S(1 + Vo)$$

And the minimum when cosine is equal to -1:

$$Pmin = S(1 - Vo)$$

The fringes will be produced with the passing of the sun above the interferometer, i.e, θ is the angle that will vary while sun is moving; because the telescope pointing will not change in this measurements. Clearing the sun visibility Vo the result is:

$$Vo = \frac{Pmax - Pmin}{Pmax + Pmin}$$

And the graphical expected result will be as follows:

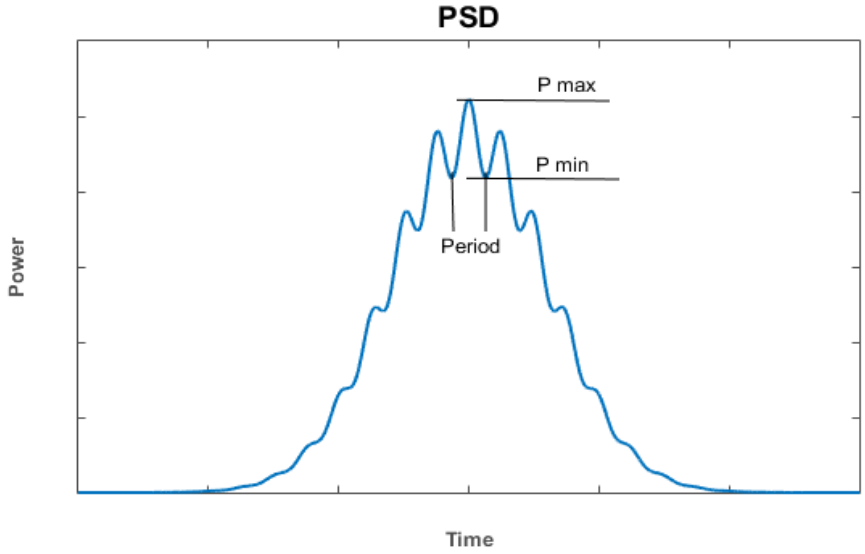


Figure 2.2.2: Expected fringes result for an extended source interferometry.

The visibility function is represented as a “sinc” function, thereby the equation to determine the sun diameter "φ” is:

$$Vo = \frac{Pmax - Pmin}{Pmax + Pmin} = sinc\left(\pi\frac{B}{\lambda}\varphi\right)$$

The second order Taylor expansion of “sinc”, gives the approximation:

$$sinc\left(\pi\frac{B}{\lambda}\varphi\right) \approx 1 - \frac{\left(\pi\frac{B}{\lambda}\varphi\right)^2}{6}$$

And finally, clearing previous equation, angular Sun diameter in radians is, approximately:

$$\varphi = \frac{\lambda\sqrt{6(1 - Vo)}}{B\pi}$$

3.- Project description

The developed interferometer catches the sky signal using two LNBs (Low Noise Blocks), which are low cost TV receptors. These need to be supplied with a DC voltage of 12V (or near) and with a sinusoidal signal. This one is requested to produce down conversion from sky frequencies to basal ones. Also, each LNB outputs the signal captured in a 75Ω of impedance connector.

The signal addition was done in a 75Ω splitter, and then all was transformed to 50Ω in order to match de input impedance of the FPGA. It was used a B200 board to capture data through a Spartan 6 FPGA, which finally, gives the digital signal to a computer in where data is processed into a PSD.

A detailed diagram of each processing block is shown in figure 3.1 and a simpler connection diagram is found in figure 3.2.

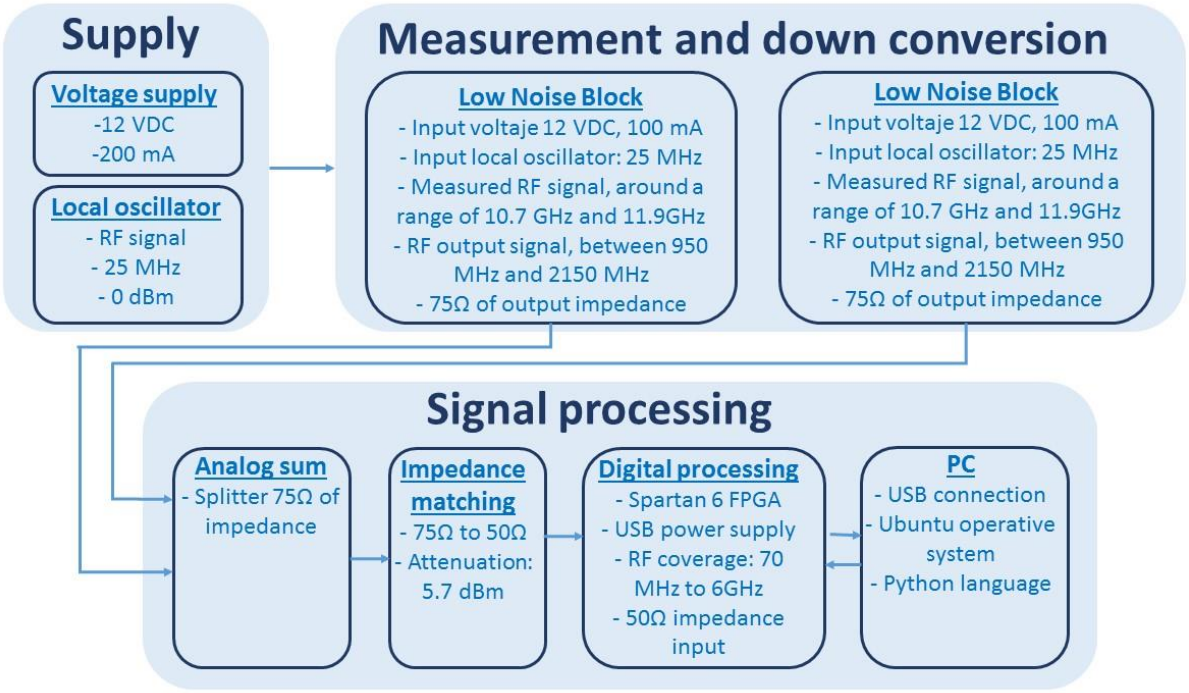


Figure 3.1: Low level block diagram of the entire interferometer

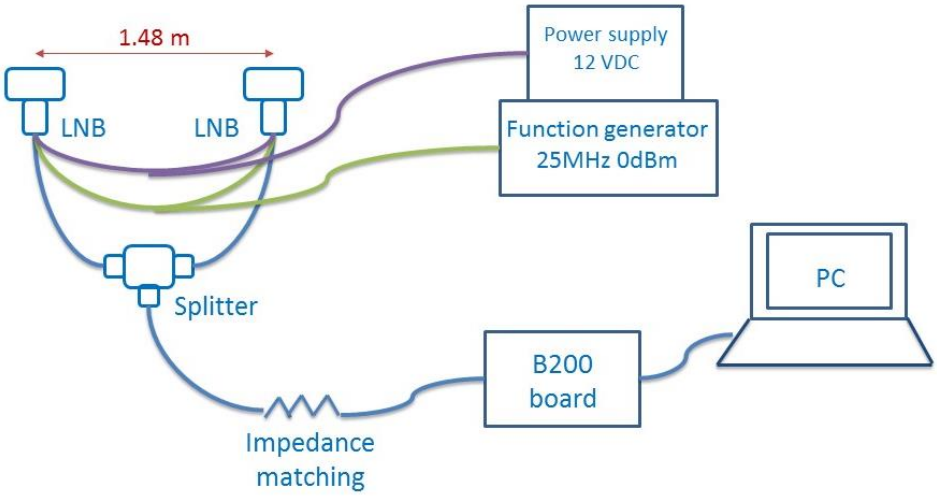


Figure 3.2: Interferometer elements diagram

As it could be notice, the project consists in two parts: hardware to capture data and software to process the information and find results. Both topics will be explained in detail in the following sections.

4.- Hardware

Each hardware elements will be explained in this section.

4.1.- Low Noise Blocks

LNBs are receptors used to capture satellite signals, which are of high frequency. For this reason, LNBs produce a down conversion step in order to obtain frequencies easily to manipulate, this ones are named “IF” (intermediate frequencies). To do the down conversion it is needed a frequency to subtract, which is given by a crystal oscillator. In this case, the oscillator was of 25MHz, but through sums and subtracts, the local oscillator achieves a down conversion of 9.75GHz (if the lower band is selected with lower supply voltage).

The output of LNBs is in a range of 950MHz and 2150MHz, however, as there are two bands, the range of the lower one is from 950MHz to 1950MHz. 12V of supply voltage places the LNB in the lower band.

Supply voltage of LNBs is given by the coaxial cable form which the output signal is obtained. To analyze the output signal and provide the 12VDC, an element to separate this signals is required. Therefore, to avoid this element (bias tee), the supply voltage was directly connected to the voltage regulator of LNB circuit, and disconnected from the coaxial output. In that case, only the RF output of LNB was traveling in the coaxial cable, and DC supply only resided in regulator and in other components that needed it. See figure 4.1.1.

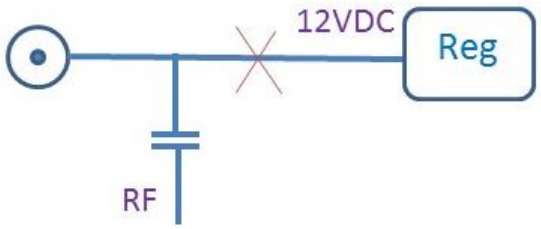


Figure 4.1.1: Separation between DC voltage supply and RF output of LNB

To make the signal from both LNBs comparable, it is necessary to have both local oscillators in phase. In other case, one frequency will be centered in different part of the spectrum because of the gap in LO phase of LNBs. Figure 4.1.2 shows this issue: a tone of 11.07GHz was captured from two LNBs with different LO source, thus the sum of the outputs reflects two peaks. If local oscillators of both LNBs were in the same phase, outputs would be centered in the same frequency, and the sum would be only one visible peak.

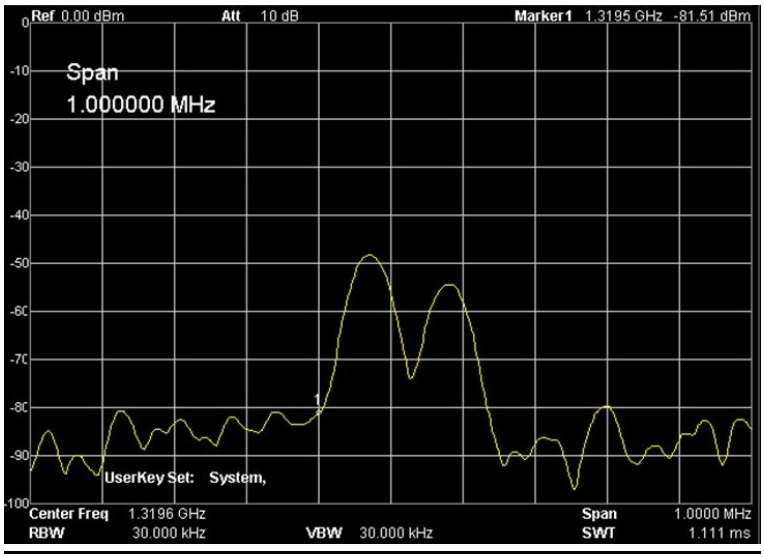


Figure 4.1.2: Addition of two LNB outputs. Each LNB had its own 25MHz local oscillator.

Both crystals were removed from LNBs and, in its terminals, a cable was joined to a 50Ω chassis connector. This is the input local oscillator; it is important to have cables of the same length to avoid producing phase gaps. That is the reason why it is not viable to use a crystal oscillator in one LNB connected to the other. This connection would have a separation equal to the distance between LNBs (1.48m) and would produce a gap in phase.

4.2.- Splitter

Splitter is a device with the function of replicate the input signal in each one of its outputs. However, if the inputs signals are connected to outputs, the input connector will hand over the addition of the inputs. This was corroborated with two inputs that had different center frequencies, as in figure 4.1.2.

The splitter used for the interferometer was of 75 Ω since LNBs connected to it had the same impedance.

4.3.- Impedance matching

Since B200 board has 50Ω inputs, it is necessary to transform the 75Ω impedance in order to avoid rebounds. To achieve this, input of 75Ω must see an equivalent impedance of 75Ω using the terminals of the other input as a resistance of 50Ω. The same procedure was developed for 50Ω side; the circuit was formed with resistors, as follows:

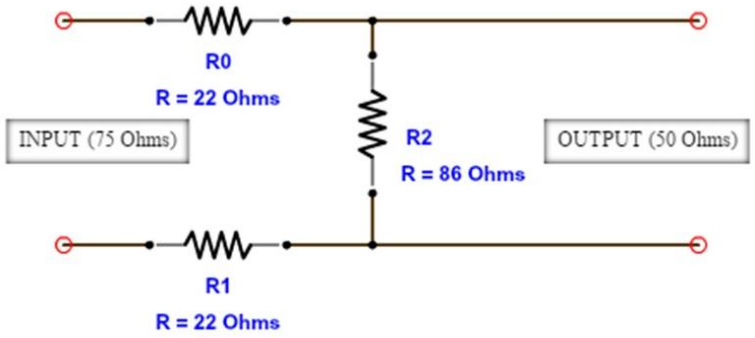


Figure 4.3.1: Impedance matching circuit

The attenuation produced by this device is around 5.7 dBm, because of the power loss in the resistances.

4.4.- Cables

This part of the instrument plays an important role because the path that signals follow must be the same from both LNBs. The phase of the signals needs to be consistent to have a sum of two comparable inputs. Thus, the RF cables (and others) were cut equals in the more precise way it was possible.

4.5.- B200 board

B200 board is a USRP (Universal Software Radio Peripheral) device which is specially designed to work with RF signals and software radio. It can be used through a computer with a USB connection, which acts like supply voltage and data channel. This board is based on a Spartan 6 FPGA, with a RF coverage of 70MHz to 6GHz, and 56MHz of bandwidth. It has one RF output (TX/RX) and two inputs: RX2 and TX/RX connectors, the last one can switch between input and output. This time, the connector RX2 was used as input of the board, leaving TX/RX as a port to obtain a possible local oscillator.

4.5.1- B200 limits - Frequency

The frequency of the local oscillator required by the LNBs is 25MHz, which is a value out of the range of B200 coverage. For that reason, it is not possible produce this oscillation with the FPGA. However, it was intended to get that frequency, and it was found that 34MHz is the minimum value B200 board allows to produce. Nevertheless, at this frequency, the output presents a lot of other peaks in spectra that makes the signal very noisy. The result in oscilloscope is not a clear wave and it probably will not work as a local oscillator to LNBs. Figure 4.5.1.1 shows the spectra of the signal produced when 34MHz are set.

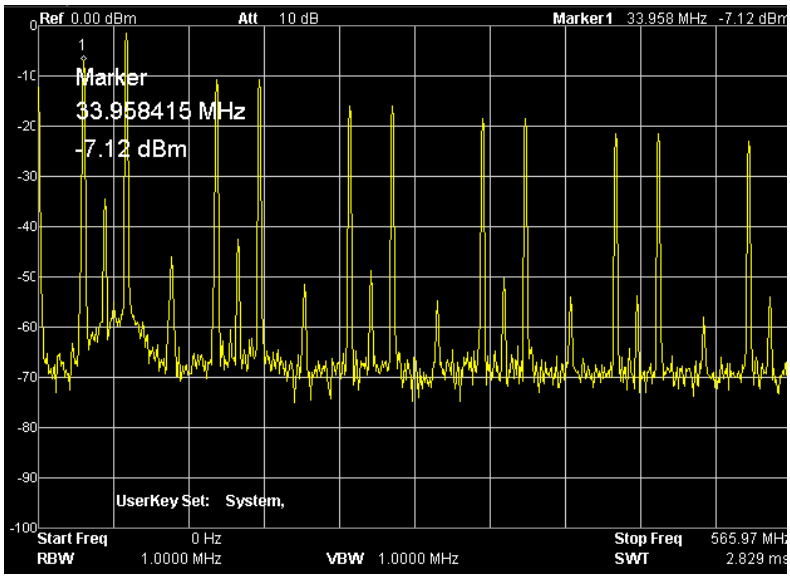


Figure 4.5.1.1: Spectra of a signal produced from USRP at 34MHz. It can be noticed that the frequency of 33.96MHz is produced, but also, a set of other frequencies.

However, a clear signal of 50MHz is shown in oscilloscope when producing a sinusoidal wave from USRP. The signal at 50MHz is produced as long as the gain and level of the signal was relatively high (compared within the range allowed, 89.5dB for transmission and 73dB for reception). Thus, the minimum frequency of a clear output signal is 50MHz, instead of the 70MHz given by B200 specifications.

All this values can be tested through an example of the library GNU Radio: “uhd_siggen_gui”. Once it is executed, waveform, gain and frequency parameters of the output RF signal can be set. However, the frequency of the waveform is not the real one which is outputted, instead, the frequency that USRP will give is the “center frequency”. When changing the value of center frequency, the output signal changes its frequency too. Also, there are some other frequency parameters: RF frequency which translates from RF and IF (intermediate frequency) and is the real output frequency; and DSP frequency which translates between IF and baseband, this one reflects the error in frequency which appears for not being able to reach the requested center frequency. For example, if 34MHz of center frequency are requested, RF frequency is 50MHz since is the minimum value which will not have problems; and DSP frequency is -16MHz, since it is the difference between 50MHz and the 34MHz requested. If a littlest center frequency is set, RF and DSP values will not change, making the output the same for frequencies of less than 34MHz.

Another interesting issue related to frequency output, is a permanent response seen in oscilloscope. When no signal is produced from GUI, the output connected in the oscilloscope shows a signal of ~4ms period, thus a frequency of 250Hz is always being produced from B200.

4.5.2- B200 limits – Sample rate

Also, it was studied the sample rate that the FPGA can achieve. In GNU Radio library, each time the B200 board is not able to accomplish the requested sample rate, messages in command terminal appears. This are ´O´ for “Overrun” which means the FPGA does not process all the samples that must receive, and ‘U’ for “Underrun” that is shown when B200 is not able to produce the samples quickly enough.

To produce the signals the maximum sample rate that does not produce ‘U’ messages is at 7MHz; and to receive data the maximum value is 5MHz. At higher rates than this maximums, data will be lost and the time between samples will not be the same in every moment.

5.- Software

For the software part of the interferometer the open source library GNU Radio is used. This one is linked with Python language and it is based on C. GNU Radio provides a powerful tool to use RF hardware with software defined radios, analyze the signals and produce them. The use of software to process signals instead of analog components such as filters, mixers, etc., makes that the same hardware could be used in different applications, and this change can be easily done through software.

As GNU Radio is a free software and it can be used with relatively low cost instruments, it satisfies the project goals. All library is based on blocks that execute tasks with the data streaming or vectors. This is done with USRP data or with signal generator data, but taking in consideration what type of data is used, i.e. floats, complex, integers, etc. GNU Radio also provides simulation tools, such as signal generators and blocks that shows data in real time, like “time sink GUI”, or “FFT sink GUI”.

For people who are not settled in programming area, GNU Radio brings a graphical way to use it, called GNU Radio companion (GRC). To use this tool, it is only needed to execute in terminal: “gnuradio-companion” command. GRC is also useful for making tests, probe blocks functionalities or knowing how to make a specific block instance in Python, since documentation of GNU Radio is not entirely completed. Every time GRC flow graph is compiled, it is produced, in the same folder of the GRC file, a Python file called with the filename of GRC GUI title. This file is the translation of the GRC code, and all blocks in the flow graph are correctly instantiated there. Also, if a block is not implemented yet in the GNU Radio environment, it could be written as an Out-Of-Tree block in C or Python languages.

For installing GNU Radio it was used a computer with Ubuntu operative system, instructions to do it are available in GNU Radio website.

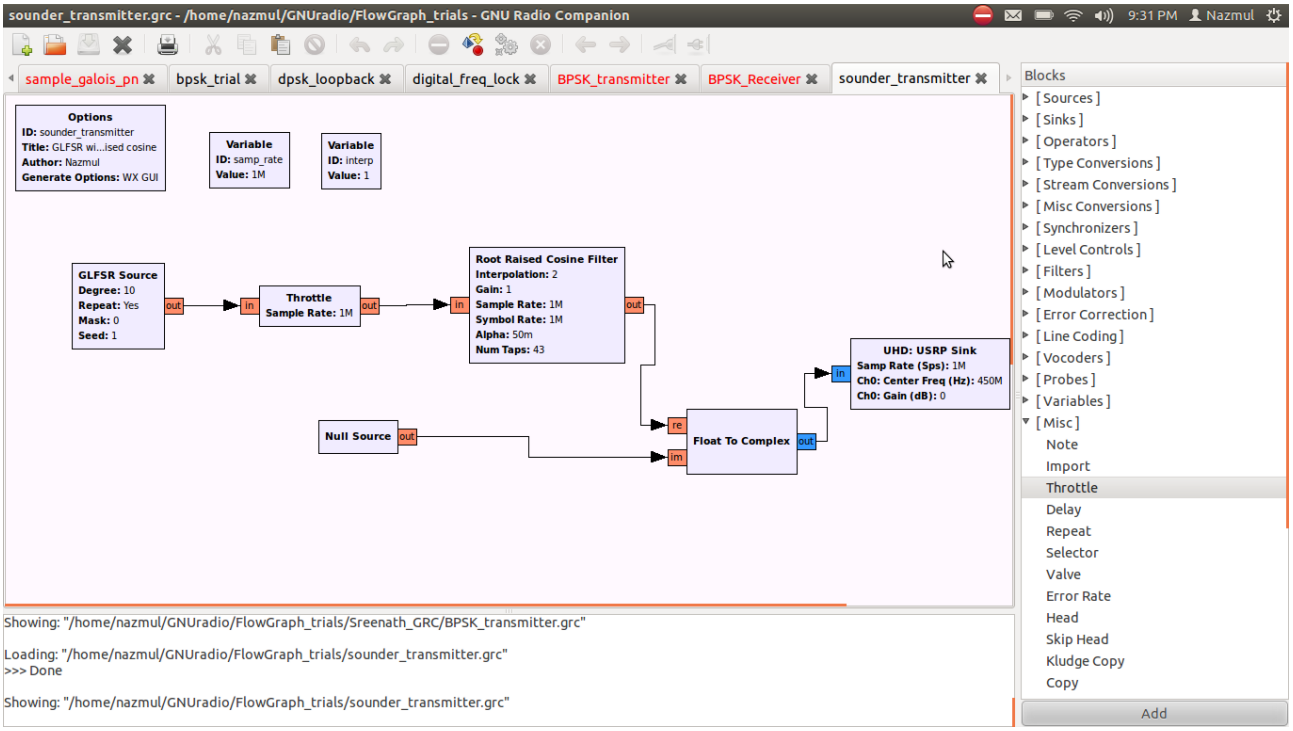


Figure 5.1: GRC GUI example. Taken from https://lists.gnu.org

5.1- GNU Radio notes

5.1.1.- GNU Radio useful tools

The library has some examples previously written in Python and GRC, this examples are ready to use and provides an element to understand the GNU Radio logic. The following list contains some of the commands that can be edited and executed in terminal once GNU Radio is installed.

uhd_find_devices: This command finds the USRP board and outputs the information of the FPGA. It shows if the B200 board is properly connected.

uhd_fft: Produces a graphical interface with the data captured through FPGA. Some parameters are editable. There are three options for seeing data: frequency spectrum, time domain or waterfall. Add the options: "- S" for oscilloscope, and "- W" for waterfall; Fourier transform is the default window.

uhd_siggen_gui: This is an example of signal generator GUI, it produces different waveforms in the range of frequency previously discussed.

gr_plot filename: When data is acquired from USRP source it is stored in IQ format, which is read by this command. It shows in time and in frequency domain the data captured in the file called "filename". If the sample rate is a known parameter, it can be adjusted using "– s" option to obtain axis values properly.

gnuradio-companion: Displays the GRC GUI to program the flow graph using graphical blocks.

5.1.2.- Definitions

In this section it will be listed some definitions that are important to comprehend GNU Radio functionality.

Data Type: Each block of GNU Radio has definite input and output data types. To connect different blocks is strictly necessary to match this data type between the ports of blocks. Data type are separated in: complex, floats, integer, short and bytes.

In addition, it is important to match the size of the items, which can be written in vector form that has a definite number of elements; or written in a stream of data, which elements are passed one by one in time, just like the USRP object outputs them.

The definition of "item" refers to one element that is passed from one block to another, an item can be a single value of a stream of data, or a vector of a given length. Each vector will be passed to the second block like a streaming of vectors.

All data types has its definite number of information, and that is a value each block needs to know. The line "gr.sizeof_type" gives it, where type refers to: complex, float, etc. If the data is a vector, the number of elements that the block needs to know is "gr.sizeof_type*(vector size)", where "vector size" is the length of the vector which passes through the input or output of the block.

Final output data (from file_sink): The output is written in a text file which contains binary data. To read the information it is necessary to consider that it is little endian, i.e. the least significant byte is in lower address.

Also, when the data is complex type, information is stored in IQ format. This means that two values per sample are stored, I and Q:

$$I = A\cos\varphi$$
$$Q = A\sin\varphi$$

Where A is the amplitude of the vector and φ the phase; I and Q are the vector component decomposition.

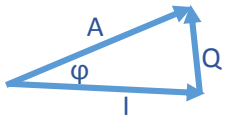


Figure 5.1.2.1: IQ data decomposition

Flowgraph: The flowgraph refers to the set of blocks which are executed in certain order. Data streaming will pass through blocks depending on the connections between them. Also, all GNU Radio program is a flow graph, because it is the central algorithm of the program, called directly from principal Python main.

Out of tree modules (OOT): GNU Radio has a great quantity of blocks to process data streaming, however, if there are some functionalities that are not implemented yet, there exist the possibility to the users to write new ones. This blocks are called out of tree modules, since they are defined within a new module. The languages used to develop new blocks are C and Python, but C modules are more efficient that Python ones.

There is a tutorial to write your own blocks in GNU Radio webpage, and it is based on the gnuradio mod-tool. When that command is called, the module and blocks files are produced in the module folder; they has to be edited to do the function required by user. It is easy edit them since files comes with marks such as "<+ +>" to complete with data types, number of ports, etc.

It is important corroborate that the new module is working properly, hence, there are QA tests, for Python an C. This tests must be written as same as the block file, and then run it to test the blocks. If QA test are passed, the block file is rightly done.

However, once it was tried to write a new module, it was hard to make it work. QA tests failed, and it was difficult to find errors. Thus it was preferred to use GNU Radio blocks and combine them to get the different processing stages.

Sink and source blocks: All source blocks corresponds to the element which provides data, which can be a signal generator, a USRP object, etc. On the contrary, sink blocks are specifically for capturing all data from flowgraph. They are graphical interfaces, data savers, among others.

5.1.3 Programming in Python

GNU Radio library has a specific way for programming, where the most important issue is to comprehend the blocks connections. Each block must be connected to others, following the processing schema according to the application. The data streaming will follow the flow graph in order, until some command stops it.

The first step is importing the modules which the blocks that will be used belongs to. Most of the modules are part of the GNU Radio library "gnuradio", but there are some others that are sub-modules of these gnuradio ones. It is important knowing the hierarchy of GNU Radio, which is available in https://gnuradio.org/doc/doxygen . For example, to import the "gr" module from "gnuradio" it must be written, at the beginning of the document:

from gnuradio import gr

Typing this line, allows to use methods and variables of module "gr", such as "gr.sizeof_complex()", among others. If the blocks are part of a submodule, they are imported similarly to the following the example:

from gnuradio.wxgui import stdgui2

The second step is related to the instantiation of blocks. Once the containers modules are imported to the actual python code, the elements or blocks which are part of its hierarchy are available to be used. Then, it is necessary create the block instances that will be connected in the flowgraph. Parameters for doing this task are different for each block and they are found in "doxygen" page (https://gnuradio.org/doc/doxygen) or in "sphinx" page (http://gnuradio.org/doc/sphinx/) which contains all GNU Radio documentation.

The connection of blocks is done through an object of "gr" class. However, since the classes corresponding to the main loop are usually created within the hierarchy of "gr.top_block", it is not necessary to instantiate a "gr" object, because the method is inherited. The command to do the connection is called "connect()" and within the "gr.top_block" class is used as:

self.connect(self.usrp_source, self.file_sink)

This line connects a previously defined blocks usrp_source and file_sink, in the given order; i.e. output port of usrp_source is connected with input port of file_sink.

If the connection is required outside the class, the method is called from a "gr" object:

gr_obj.connect(usrp_source, file_sink)

Where "gr_obj" is a previously defined gr class object.

It is important to know that each block is not only joined with only one block, instead it can be connected to a set of other ports. Thus, the signal obtained from one block output is copied to various input block ports. Also, there are some blocks which have various inputs and outputs; as it are different signals, have different port number. As an example, the add block has two or more input ports, which are different signals. In the following lines, the add block is connected to sink1 and sink2 blocks (previously instantiated), and the only one output signal of addition (sink1+sink2) is connected to the multiply block and the file sink.

self.connect(self.sink1, (self.add_block,0))

self.connect(self.sink2, (self.add_block,1))

self.connect(self.add_block, (self.mult_block,0))

self.connect(self.add_block, (self.file_sink1,0))

The ports of each block are written as a tuple, where the first output is named by "0" and the others are named with the successively integers. As it can be noticed in the third line, a block that have only one input (mult_block multiplies the stream of data by a constant, thus receives one stream of data) is written with its explicit port; but it also could be omitted:

self.connect(self.add_block, self.mult_block)

The same rules applies to input or output ports. In addition, the command "connect" also admit connecting a series of blocks in one line:

self.connect(self.usrp_source, self.mult_block, self.file_sink)

This is possible whenever the blocks in the list only have one output port, because the command connects the blocks in a sequence.

As said before, the blocks must be defined within a class in order to create a flow graph which contains the steps given by the connections. Hence, the class is created in the "gr" module hierarchy, as the example:

```
class my_top_block(stdgui2.std_top_block)
    def __init__(self, param1, param2, …)
        stdgui2.std_top_block.__init__(self, param1, … )
```

The class "my_top_block" is part of the class "std_top_block" which, at the same time, is part of "stdgui2" module. All the flow graph will be indented at the level of "def __init__" since that method is the one that is executed at first time when instancing "my_top_block" class. The third line only initializes the "*stdgui2.std_top_block*" class using the wanted parameters values. It is important to have in consideration that the parameters of the definition of "my_top_block" and "*stdgui2.std_top_block*" are not necessarily the same; the principal class could receive more parameters.

The module "*stdgui2.std_top_block*" is part of "*gr.top_block*", but the first one helps to build a graphical interface. However, if flowgraph classes are defined based on "*gr.top_block"* module, it will work as the same as defining it based on "*stdgui2.std_top_block*".

The principal "main" of python code must instantiate the class "my_top_block" to make the flowgraph run, stop, or behave as it is expected. Thus, the main, at the end of the text document, should look as:

```
if __name__== ´_main_’:
    try:
        my_tb = stdgui2.stdapp(my_top_block, "Title", nstatus=1)
        my_tb().run()
```

Where "run()" is a method of the "*top_block*" module. This one makes the flowgraph runs the algorithm. There are other methods that are useful to control the class instance, such as:

run(): It is the simplest way to run a flowgraph. It calls "start()" and then "wait()". This method will stop once the flowgraph ends or when other method calls "stop()", but it does not need to be stopped by code.

start(): Starts the flowgraph, but does not execute the same procedure as "run()": "start()" only makes the graph flow.

stop(): Stops the flowgraph.

wait(): Waits for the end of the flowgraph. It should be placed before "stop()" to make sure the flowgraph complete all its tasks.

lock(): Lock the flowgraph, for making a new configuration of the blocks (connect or disconnect, add new blocks, among others)

unlock(): Makes the flowgraph run again. Has to be called after a "lock()" method.

The usage of this methods should control the flowgraph properly, and if the previous instructions are followed, it would be easy to write an algorithm in Python (as the same as GRC). If there are errors, the usage of each block could be find in documentation, or easier, in GNU Radio Companion, by compiling a working flowgraph. Also, the blocks can be defined in an interactive Python shell, such as "Ipython". That is why it was important to comprehend from which class the connection of blocks is done, since it is easier to avoid class definition for making tests in Ipython. In other words, it is more practical to use "*gr_obj.connect(…..)"* directly. Besides, Python shells allow to define object of different classes and obtain a list of its methods by pressing "tab" key. This last tool is very helpful because GNU Radio documentation is not very extensive and, since blocks inherit methods of the parent classes, it is easier to know all the list of possible methods.

If the previous tools fails, probably, the error is related to a bad connection: some port unconnected, data type mismatched or inconsistent length of vectors. This kind of errors will probably output in terminal: "unable to coerce endpoint" message. The only way to repair it is looking, meticulously, at all blocks connections.

5.1.4 Blocks and modules

This section contains a list of blocks which were used to obtain the interferometer data or that are useful in the testing of GNU Radio. It is important having in mind that the blocks names commonly gives information about the input and output types. Thereby, a block name such as "block_abc" has an input of type "a", output of "b" and the inside steps of the block are done in "c" type. That values are "f" for floats, "c" for complex floats, "i" for integers, "s" for shorts and "b" for bits. A "v" in the first place means that the blocks operates with vectors. For example: "nlog10_ff" has for input and output float types, and "fft_vfc" has float inputs, complex outputs in vector streaming.

add_vff: Sums two or more vectors, element by element. Operates with vectors of floats in its input and output. This block will only outputs data when all inputs has a vector copied on it.

complex_to_mag_squared: Calculates the square of the magnitude of a complex number. Input corresponds to a stream of complex data and output is float type.

deinterleave: Separates the input stream of data in a given number of outputs. The separation is done in a certain number of elements, given in "blocks". If the block size is n, the first n elements are passed to the first port, the second n elements to the second port, and successively until complete the quantity of outputs. Then, the separation will starts in the first output again.

fft_vfc: Calculates the Fast Fourier Transform of each vector of floats that enters in the block. It outputs a complex vector containing the FFT.

file_sink: It will store all data coming from the previous block in a file (of the given filename). The format of the data can be of any type and length, file sink will store them in binary. If type is "floats" then, binary data will be in little endian format. Also, file sink will store data all time the block is connected, thus if it is connected a wide period of time, files will be larger and will contain more samples. However, binary size is lower than other formats which is a great advantage. The flow of data can be controlled by "head" or "valve" blocks.

head: This block takes a given number of samples a copies them into its output. However, when using this block all flowgraph stops; thus, it is necessary reconfigure flowgraph every time "head" is used. For example, to save a set of data in a file sink, the head block is connected to data source and sink; but for making the graph flow again, the head block will have to be disconnected. This is achieved with the "lock()" and "unlock()" commands from "gr".

integrate_ff: This block takes a given number of floats samples and sums it. "integrate_ff" gives a float number for each set of samples it sums, thus it makes a sub-sampling of data. As its name says, it receives a float data streaming.

message_sink: Receives any type of data, since all streaming has messages. Messages are the units GNU Radio works with, and this sink allows user to manipulate them. However, it is difficult to obtain data from message sink since it has to be called from outside the flow graph, thus this data cannot be used in the algorithm itself. In the developed software message sink is only used for receiving data and producing the GUI.

multiply_const_vff: Multiplies a constant vector by the input vector of floats. It outputs a float vector too.

nlog10_ff: Calculates the base 10 logarithm of input floats and then multiplies each value by n. It gives floats numbers to the following port.

null_sink: Since it is necessary to connect all blocks, when it is needed to test a flow graph, this block can be used to give an auxiliary input port. "null_sink" does not execute any command, just receives data and do nothing with it.

null_source: As the same of "null_sink", this block supplies any port that needs to be connected. The output of this block only contains a stream of zeros that can be connected to any block.

stream_to_vector_decimator: Produces a vector of a given number of samples of the input stream of data. Thus, it takes a set of samples and produces one item with it, the item is a vector of length equal to the number of taken samples. That is why this block is called a "decimator", because it makes a sub-sampling of the streaming rate.

throttle_block: Makes a sampling of data for avoiding take all computer memory. Only use this block for simulation tasks (it is very necessary to use it) and never with hardware. Using it with hardware will interfere with the sample rate given by it.

usrp_source: It is one of the most important blocks, since it makes the connection between hardware and GNU Radio. This object produces a stream of data captured from USRP (in this case, B200) using a set of parameters given by user.

valve: Through a Boolean instance, it controls the flow of data between two blocks. When it is opened data does not pass through this block, thus it does not arrive to the following one. It is useful to control the streaming stored in a file sink.

vector_to_stream: Translates each received vector into a stream of data; hence, output will be a sequence of items of any type.

5.1.5 USRP and FFT parameters

One important issue to consider is the definition of parameters. USRP_source from uhd (USRP hardware driver) class has tools to define the USRP parameters values, such as set_gain(), set_center_frequency(), etc. Some of the parameters are listed here:

Gain: The gain applied to the data acquired from hardware. It is measured in dB, and the limits are: 89.5dB for transmission and 73dB for reception.

Center frequency: Is the frequency at which the bandwidth is centered. It is an important value at the time of making measurements because the hardware only takes a small window in frequency, thus it is important the correct tuning of this value.

DSP frequency: Translates between IF and baseband frequency. It is the error of the requested frequency and the real output frequency of the signal generator.

RF frequency: Translates RF frequency to intermediate frequency (IF). In signal generator program, it corresponds to the final value of the output frequency in USRP.

Sample rate: It is the number of samples per seconds that the B200 board consumes or produces. It is also, the bandwidth of the FFT of data. When changing the sample rate, it also will change the bandwidth of the acquired data.

Bandwidth: It is expected that bandwidth was the range of FFT result, however, that value is given by sample rate (no matters Nyquist theorem). Bandwidth parameters is a sort of filter in data acquisition which multiplies by zero all values out of this range, centered in "center frequency". Hence, bandwidth is a passband of the data, but is not the window of observation. For that reason bandwidth and sample rate were set in the same value for having a minor loss of data.

FFT size: It is the number of bins of FFT, it can be changed in the "uhd_fft" example, as the same as in the developed software. It is a FFT block parameter.

FFT rate: Counts how many FFTs are updated in time. A fft rate of value= 30, means that it will be produced 30 FFTs in a second, approximately. It is a FFT block parameter.

5.2.- Data acquisition

Acquisition was developed through GNU Radio in a Python file. Data was captured from a USRP object, resampled to get the FFT rate parameter and then processed. It was first calculated the Fourier transform, and then, the square of the module. This produces a given quantity of FFTs in time (fft rate), but since the average of some samples of FFT was requested, this time is increased. The mean data of FFTs was saved in a file, but it was also putted in a graphical interface. Thus, it was calculated the logarithm of the data to be able to see it in dBm. Finally, it was calculated the data that was important, the PSD (Power Spectral Density), which was the integration of each FFT average. This values in time were also stored in a file sink. See figure 5.2.1 to have a graphical vision of the algorithm.

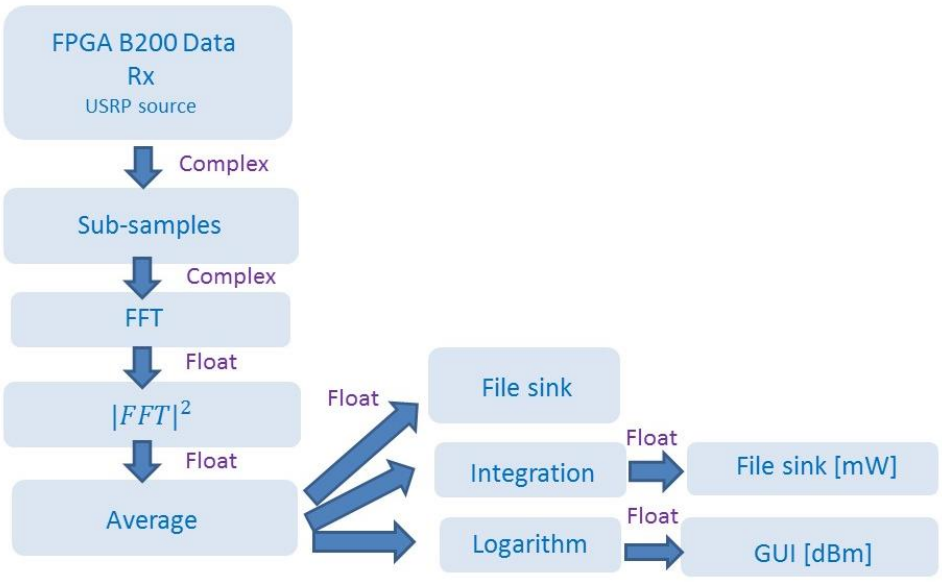


Figure 5.2.1: Summary of the data acquisition procedure

For each one of the tasks detailed before, it was used certainly blocks (which functions was defined in previous section). The flow diagram is shown in figure 5.2.2, that also includes a save data button. When this check box is pressed, flow graph stops for some seconds because of a "head" block. Then, it stores a given number of samples in a file with the user given filename. For achieving this, it was created the needed blocks and stopped all flow graph. Later on, it was

connected the head and file sink blocks during some seconds, to then, make the same procedure for reconnecting the scope sink (the “No” branch in figure 5.2.2). If the data in GUI is being studied or saved in the “integration_data” file it is not a good idea to press this check box (save data), since “head” block will stop the entirely flow graph.

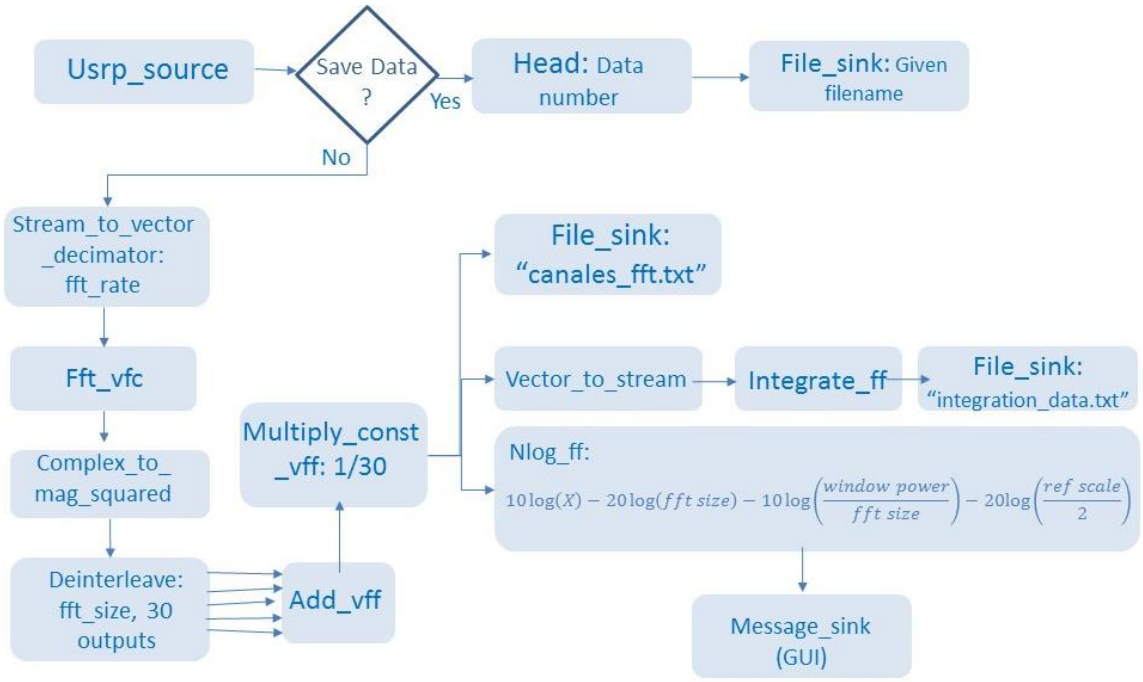


Figure 5.2.2: Flow diagram using real GNU Radio blocks

The average procedure was done with 30 samples, thus, if FFT rate is 30 FFTs per second, each point of PSD will be produced in a period of one second. This is the value that will be used in both, data acquisition and offline data processing. On the other hand, the block nlog_ff, was initialized with n=10, for obtaining the dBm calculations, and it was summed to that value the set of constants given in figure 5.2.2 in order to adjust the levels of the signal in the screen.

All other parameters are adjustable in the GUI (see figure 5.2.3) produced by “uhd_fft__psd” which is the Python file that stores data and calculates PSD.

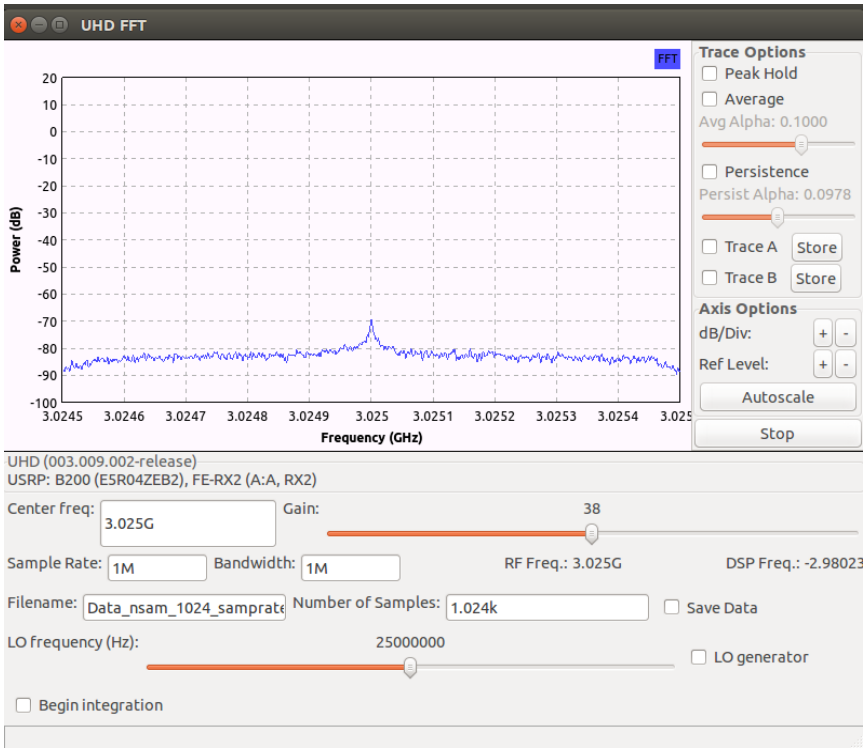


Figure 5.2.3: GUI of “uhd_fft_psd” data acquirer

Once this GUI is opened it begins saving the PSD data. It was tried to use a valve controlled by a button to start and stop the acquisition. However, some errors appeared, and it were not resolved.

GUI also provides a check button for generate a sinusoidal wave, but as it was developed in section 4.5, it is only possible to generate signals of more than 50MHz, with a great gain (parameter that was not adjusted in uhd_fft_psd).

5.3.- Offline data processing

All the Sun diameter calculations are done after the data acquisition, thus, it is an offline procedure. The equations of the second section of this document were translated into code, which is only based on Python (no need to use GNU Radio, since binary data is easily read). Also a GUI was added through TKinter library of Python to interact with user.

The user will be firstly asked to open the file that contains data, which is “integration_data.txt” if the filename was not changed from “uhd_fft_psd”. Then, all parameters has to be adjusted, following the given units and instructions of GUI. Once it is done, user must press “Set parameters and plot data” in order to show the data obtained in the measurement. It is done after setting parameters because it is necessary to calculate the time axis to plot it. When the data is in screen, “click to input points” has to be pressed and then GUI will ask for the maximum and minimum power, and two points that marks a period. Then “Compute” will calculate baseline and the sun diameter with the given data.

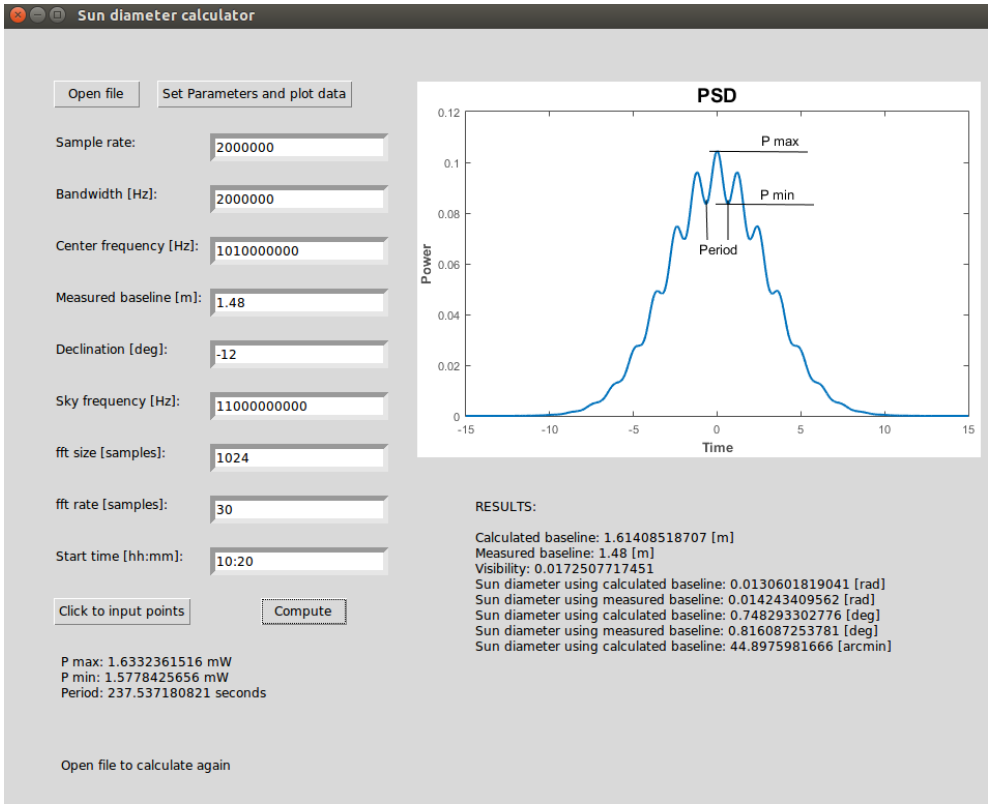


Figure 5.3.1: Sun diameter calculator GUI

Python file which opens this GUI is “offline_proc_gui”, however, a simpler code was implemented: “offline_proc”. This last file does not open a graphical interface, but instead, it is easy to read and understand calculations. If user studies the Python code, he will find that some of the initial samples are omitted; the reason is based on the data acquisition method, since the measurement is done at the time GUI is opened; thus, it is necessary a time (a set of samples) to adjust parameters. Hence, this first samples are disturbed by a change in parameters and are not significant.

6.- Usage

6.1.- Hardware

Once all hardware is mounted, the interferometer must be aligned in East-West direction in order to have the Sun passing in different angles above each LNB. This will produce fringes, because if it aligned in North- South direction the Sun will pass in the same angle above LNBs along the path. Also, if the interferometer is not aligned, the baseline will decrement to the longitude equal to the projection of interferometer baseline in East-West direction (since Sun travels along this direction due Earth rotation).

The maximum power of the sun is produced when it is at the maximum elevation that can achieve depending on the geographical location. Hence, it was investigated the hour of the day at which the sun was near to the zenith. It was at 13:50 hr. at an elevation of approximately 78°, according to the day date. As the fringes period at a baseline of 1.48 m is, approximately, 4 minutes; it is not necessary to integrate a great period of time. Some tools for the interferometer alignment are listed in bibliography section.

The experiment acquired data along a few hours (but it is not necessary, it just has to measure a pair of fringes), before and after 13:50 hrs. Different parameters were tested, at a center frequency of 1.01GHz since it was one of the points in spectrum which had higher power values. However, all spectra power changed almost the same quantity when covering and uncovering the sun radiation; thus, it is not a definitely final value for center frequency.

Figure 6.1.1 shows the real prototype, capturing data from Sun.

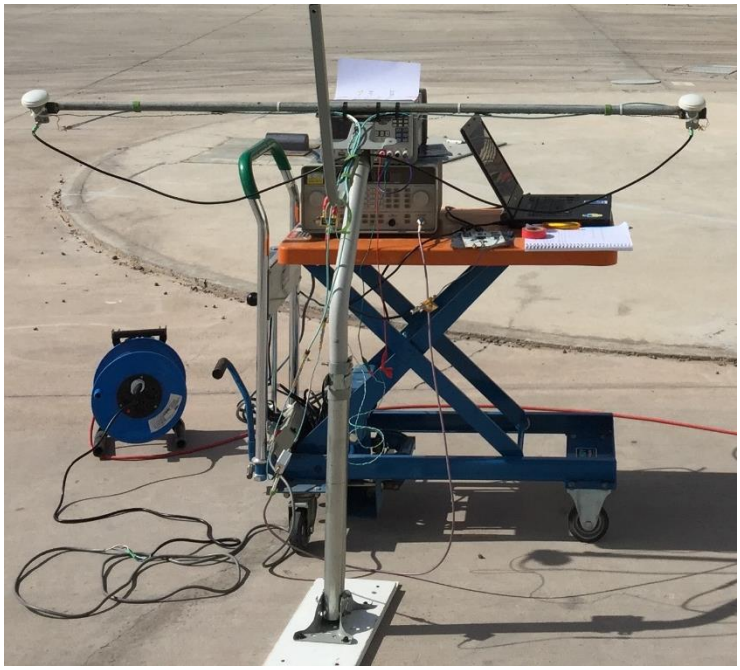


Figure 6.1.1: Interferometer hardware setup

6.2.- Software

To capture data with “uhd_fft_psd” it is necessary to provide the other files and modules which contains all blocks that will be used. This files are:

- fftsink2.py which imports the GUI and FFT modules
- fftsink_gl.py which creates blocks to process data, such as integrate, vector to stream, file sink, etc.
- logpwrfft.py.txt which instantiates FFT and logarithm blocks

All this files comes with GNU Radio installation, but the files in the repository are an edited copy of them. Therefore, user must find these files in GNU Radio folders and replace them with these new codes. It is a good practice to conserve GNU Radio original files with other filename, for example “fftsink2_original.py”, or change the names of these new flowgraphs and change it too in all modules that are using them. It is important to know that GNU Radio has two places with the same files, the one that is only accessible by administrator contains the files that has to be changed. However, if you do not want to modify that files, the new files could be added to the path of Python for making a clearest use of GNU Radio.

Once done the previous steps, execute “uhd_fft_psd” in the folder you want to save data. Information captured from USRP will be stored at the time the GUI is opened.

As “offline_proc_gui” does not use GNU Radio, it does not need a special folder to work, but it needs the image “grafica.png”. This figure is shown in GUI just to compare it with experimental data. All this files are stored in a GitHub repository: https://github.com/ldauvin/Two-Station-Interferometer.git

7.- Results

The collected data produced a fringe pattern that was comparable with the theoretical period. However, it is a very noisy pattern which has great number of oscillations in high frequency. Also, the amplitude of the entire signal does not follow the expected curve. An example of the results is shown in figure 7.1.

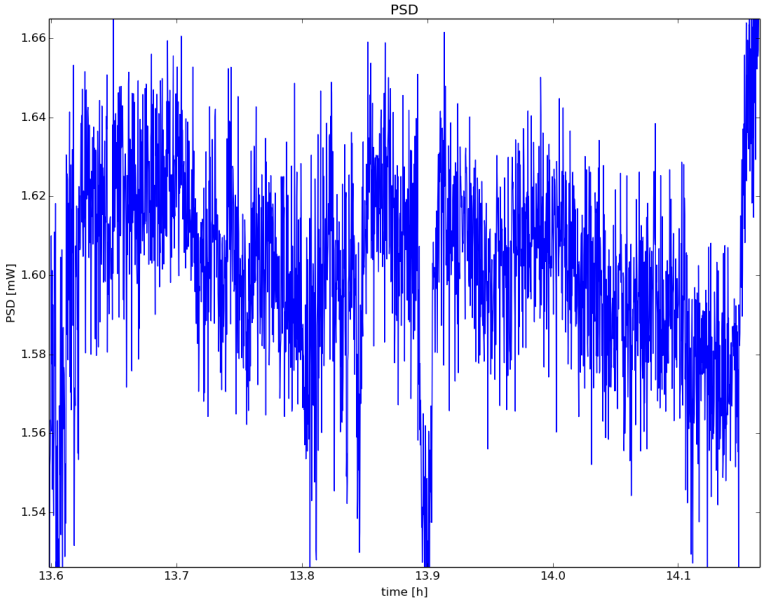


Figure 7.1: Fringe pattern obtained with the interferometer

Figure 7.1 shows less than an hour of integration, but fringes are recognized in around 4 minutes, as expected. This little sector of data presents a mean value relatively constant, but when a larger window of time is studied, it is found that PSD has jumps in power (See figure 7.2).

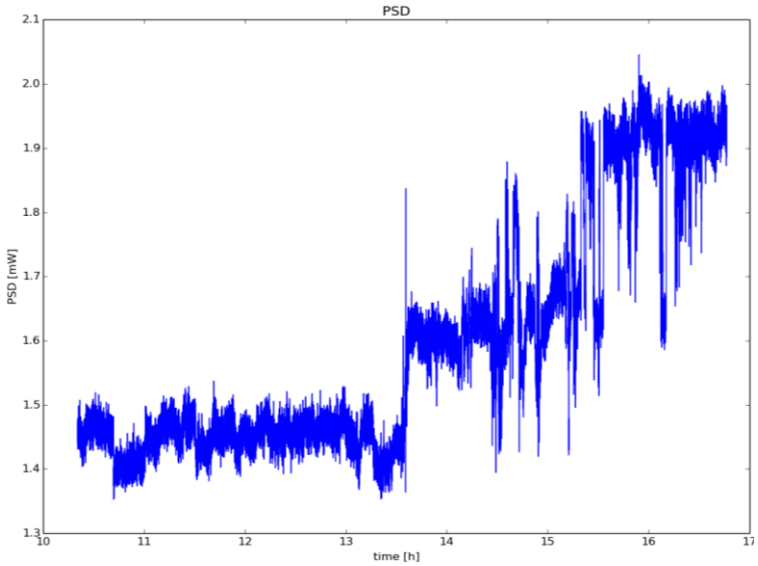


Figure 7.2: PSD captured in a long period of time. Figure 7.1 is a section of this data.

The amplitude gaps are probably produced by the LNBs gain adjust, which produces a bigger gain for low power signals and vice versa.

The data shown in figures corresponds to a measurement centered in 1.01GHz, bandwidth of 2MHz, sample rate of 2M, 38 dB of gain, a FFT of 1024 bins and a FFT rate of 30 FFT/second (which means 1 sample of PSD per second). Period of fringes is in the graph approximately 237 seconds, which leads a 1.61m baseline. Although the Sun diameter was calculated as 44.9 arcmin, the selection of the maximum and minimum power makes this result vary significantly. Hence, it is very important to have clearest measurements for taking this data points.

All other measurements produced similar results, thus they are attached in the annexes section.

8.- Cost estimation

One of the objectives of the project is achieve a low cost instrument. Thus, an initial cost estimation is presented here:

Element	Price (CLP)
LNB	2 * 4000
Splitter 75Ω	2000
Cables	5000
Resistances, connectors and other electronic components	5000
B200 board	545000
Power supply	10000
LO source	2000
Total:	**577000**

Table 8.1: Interferometer cost estimation

The previous table is only a preliminary estimation subject to prices at the present time. For realistic results, laboratory instruments were replaced in the estimation. Power supply was replaced by the cost of a 3A power supply, which is substantially cheaper. Also, the function generator was replaced by a circuit which only uses a crystal oscillator, capacitors and universal copper boards. This changes will be explained in section 9.

9.- Forward stages

Since one of the objectives of the interferometer is its low cost, it would be necessary replace the laboratory supply voltage by a low cost one, such as supply voltage of 3A (~CLP$10.000) or a computer transformer. This kind of AC-DC voltage converters should have a clear enough signal for avoiding any perturbation on the LNBs performance. The DC voltage supply of LNBs insides substantially in the functionality of it, thus, it is important the kind of DC supply which will be used. For similar reasons, the function generator must be replaced as LO signal; instead, it is a good idea evaluate a LO built from a crystal oscillator of 25MHz. It would be important that the circuit had the same length of cables to both LNBs in order to avoid phase gaps. The cost of interferometer would decrease with this modifications, and it could fulfill that project objective.

Another objective that is not complete yet, is related to the easy understanding of data for students. Although fringes are visible, the information obtained at the moment is different from ideal one. One problem is the amplitude gap due by the gain adjust of LNBs; first of all, if this adjust is not the same for both LNBs, the signals will not be comparable, thus the sum result will not make sense. Therefore, a primary step must be making sure that both LNBs are amplifying the same value at the same time. Once that is corroborated, the gain adjust could be compensated in time, because the gaps seem to be discrete: for a range of input power, gain is a given value. GNU Radio USRP block has the gain parameter, which controls gain obtained from B200.

Noise is reduced when a higher range of spectra is integrated, i.e. when the bandwidth of observation is bigger. Hence, user could manipulate this value to get better results, but, as it was commented, real bandwidth is equal to sample rate. One could think in adjusting sample rate in higher values, but the maximum allowed is 5MHz, since at greater rates overrun is produced and samples are missed. Thus, maximum bandwidth is 5MHz, which is a relatively low range (a LNB covers around 1GHz). One solution for this issue would be a frequency scan, in other words, change the center frequency to make a set of measurements that covers a higher range of the spectra. However, 5MHz is a really low value, thus the change in central frequency should be done a big number of times. Also, it could be intended to make bandwidth independent of sample rate, thus, bandwidth could increase without data loss. Nevertheless, Nyquist theorem requests a sample rate higher than two times of bandwidth. Although, an equal sample rate and bandwidth does not follows the sample theorem, it is an issue to be studied.

Maybe, the data loss could be studied if every time a sample was stored in file sink, the real time was stored too. This value would be more accurate than the approximation done until now (which depends on FFT rate and number of FFTs to make the averaging). Thereby, the time at which the samples are skipped with a higher sample rate would be known, and calculations would be more precise.

To avoid noise, data acquisition could have an average with more number of samples. Actually, there are 30 FFT averaged in one second, but with “FFT rate” user could control this time step. It would be useful to add a GUI button to change the number of FFTs to be averaged and the FFT rate, to control noise and time. Also, in the offline calculations, it could be applied a low pass filter to eliminate the higher frequencies. This would make the data clearest to the user. Further, it would be useful to have “data integration” button to control the file sink of PSD; that would avoid the first non-significant samples that are affected by the parameters adjust.

Another issue in the data noise is related to the temperature at which FPGA operated. Since the Sun is being measured, the temperature due the radiation could interfere in data acquired by B200 board. However, it is a not studied effect.

Finally, it is necessary to comment that resolution calculations were not taken in consideration, but it is important to know if the 1.48 baseline is enough to resolve a section of the sky of 30 arcmin (~ Sun angular diameter). Although the Sun diameter could not be resolved, power will come to interferometer and it will change the FFT measured amplitude. However, it would introduce some quantity of distortion if the Sun cannot be properly observed by the instrument. An important following step could be study this value.

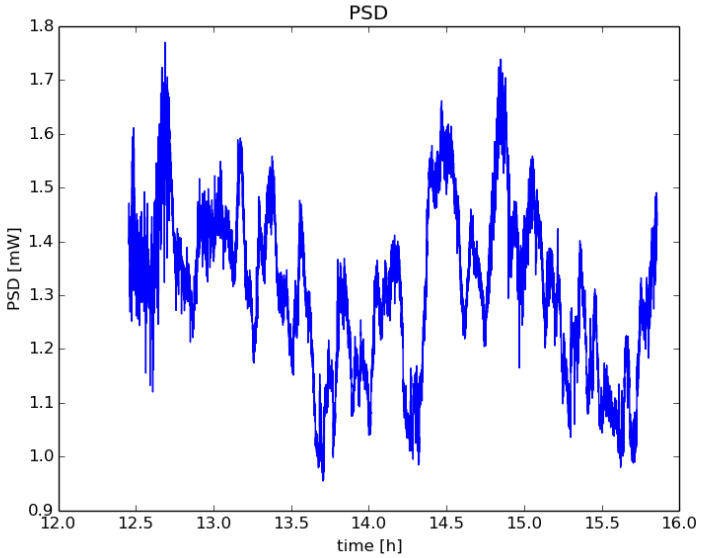
10.- Bibliography

- Doxygen. *"GNU Rradio Manual and C++ API Reference"* [on line]. [Visited on February, 2016]. Available in: https://gnuradio.org/doc/doxygen/
- Ettus Research. *"USRP B200/ B210 Information sheet"* [on line]. [Visited on February, 2016]. Available in: https://www.ettus.com/product/details/UB200-KIT
- KEITH, Daniel *"Digital Modulations Using the Universal Software Radio Peripheral"* [on line]. California Polytechnic State University. San Luis Obispo, 2011. [Visited on February, 2016]. Available in: http://digitalcommons.calpoly.edu/cgi/viewcontent.cgi?article=1105&context=eesp
- KODA, Jin et al. *"A Michelson-type radio interferometer for university education"* . American Journal of Physics, January 14, 2016.
- PV Education *"Sun Position Calculator"* [on line]. [Visited on February, 2016]. Available in: http://www.pveducation.org/pvcdrom/properties-of-sunlight/sun-position-calculator
- Sphinx. *"Gnuradio documentation"* [on line]. [Visited on February, 2016]. Available in: http://gnuradio.org/doc/sphinx/
- Timezone db *"Time Zone Database"* [on line]. [Visited on February, 2016]. Available in: http://timezonedb.com/
- VON STEINKIRCH, Marina *"On the calculation of the angular diameter of the sun by Michelson radio interferometry"* [on line]. State University of New York. [Visited on February, 2016]. Available in: http://astro.sunysb.edu/steinkirch/reviews/radio_steinkirch.pdf

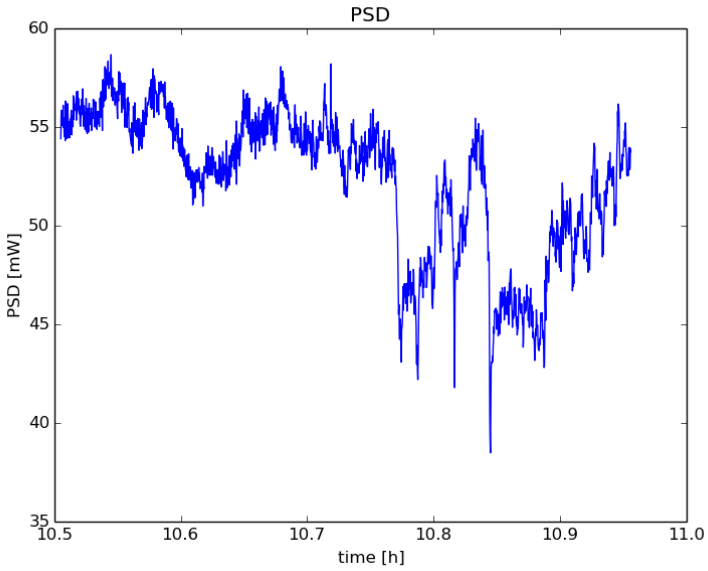
11.- Annexes

A.- Measurements and results

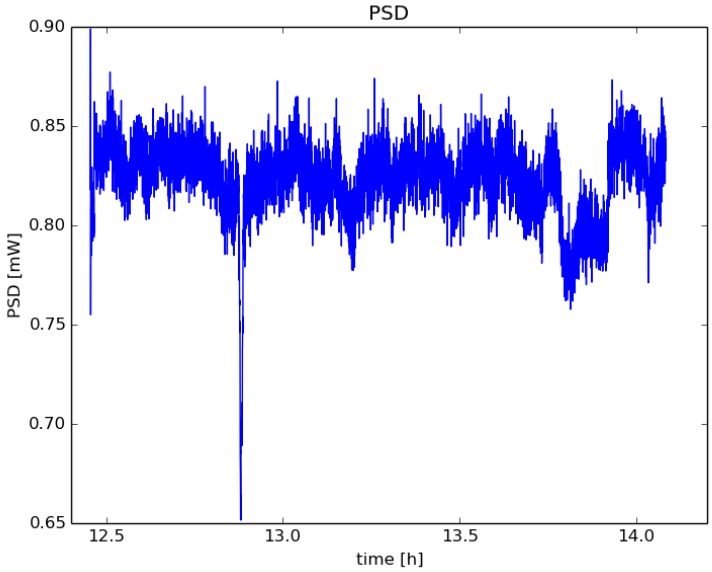
Measurement 1
B= 0.428 #Baseline [m]
freq=11e9 # Sky frequency
dec_deg=-12.27 #deg
samp_rate=1e6
fft_size=1024
fft_rate=30
start=10*3600+(30*60) #10:30 AM
filename='20160218_data.txt'



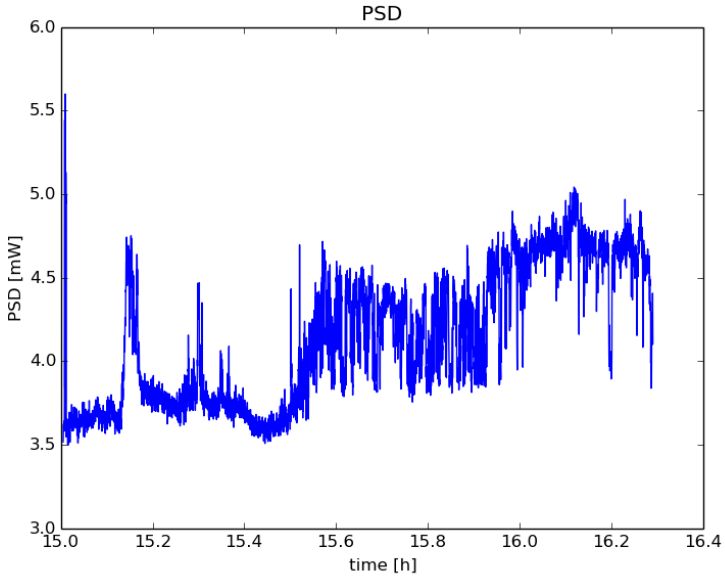
Measurement 2
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=32e6
fft_size=1024
fft_rate=30
start=10*3600+(30*60) #10:30 AM
filename='20160222_data.txt'



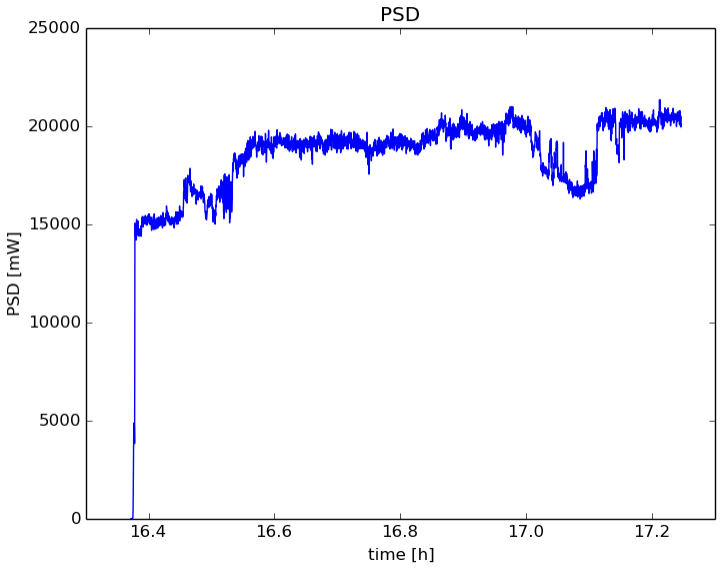
Measurement 3
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=6e6
fft_size=1024
fft_rate=30
start=12*3600+(27*60) #12:27
filename='20160223_data.txt'



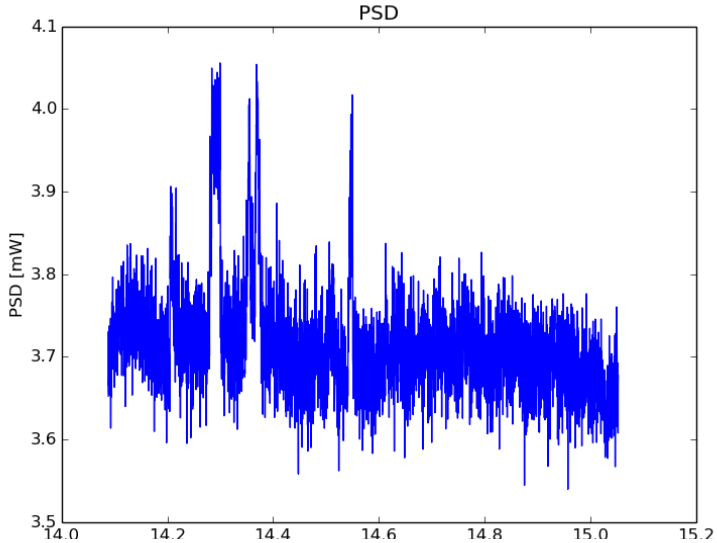
Measurement 4
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=5e6
fft_size=1024
fft_rate=30
start=15*3600+(00*60) #15:00
filename='20160223_data2.txt'



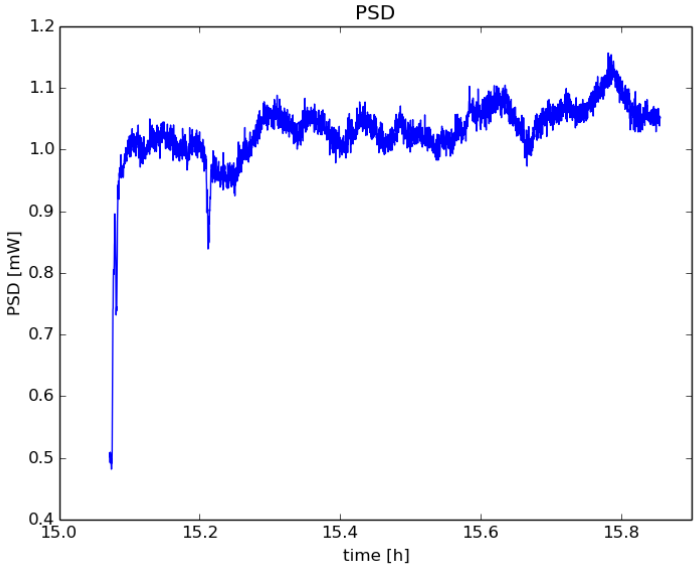
Measurement 5
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=6e6
fft_size=1024
fft_rate=30
start=16*3600+(22*60) # 16:22
filename='20160223_data3.txt'



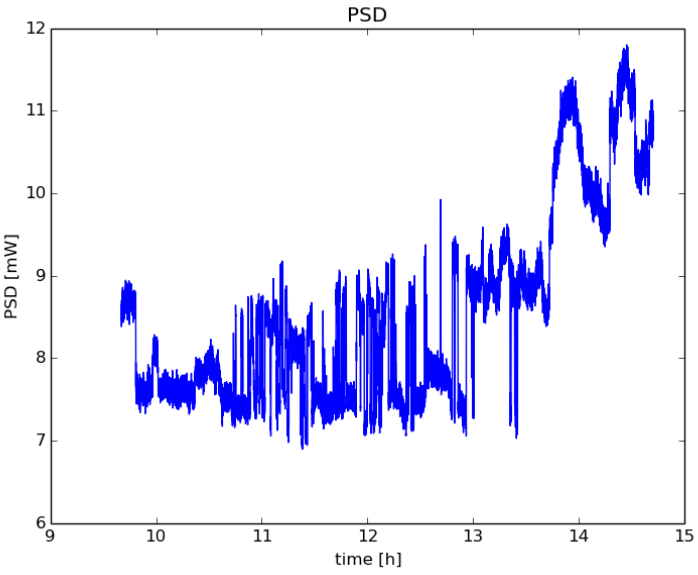
Measurement 6
BW=4e6 # RF bandwidth
gain=38
center_freq=1010e6
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=8e6
fft_size=1024
fft_rate=30
start=14*3600+(05*60) # 14:05
filename='20160224_data.txt'



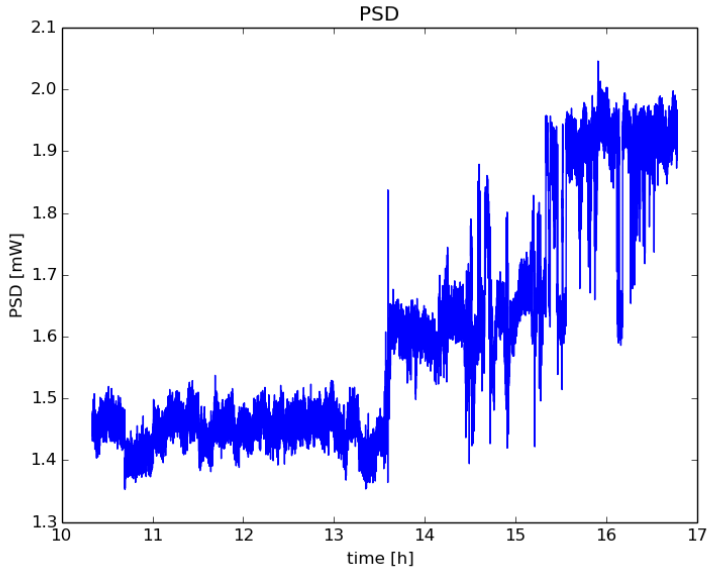
Measurement 7
#15:04 till 15:52
#integrates 48 minutes
BW=1e6 # RF bansdwidth
gain=38
center_freq=3.025e9
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=1e6
fft_size=1024
fft_rate=30
start=15*3600+(04*60) # 15:04
filename='20160224_data2.txt'



Measurement 8
# 09:40 till 14:50
BW=8e6
gain=38
center_freq=1010e6
B= 1.483 #Baseline [m]
freq=11e9 dec_deg=-12.27 #deg
samp_rate=1e6
fft_size=1024
fft_rate=30
start=9*3600+(40*60) # 09:40
filename='20160226_data.txt'



Measurement 9
# 10:20 till 16:48
BW=2e6
gain= 38
center_freq=1010e6
B= 1.483 #Baseline [m]
freq=11e9
dec_deg=-12.27 #deg
samp_rate=2e6
fft_size=1024
fft_rate=30
start=10*3600+(20*60) # 10:20
filename='20160227_data.txt'



Measurement 10
# 10:41 till 19:00
BW=5e6
gain= 38
center_freq=1010e6
B= 1.483 #Baseline [m]
freq=11e9 dec_deg=-12.27 #deg
samp_rate=5e6
fft_size=1024
fft_rate=30
start=10*3600+(41*60) # 10:41
filename='20160228_data.txt'