Understanding Digital Circuits

Leonardo Sattler Cassara, e-mail:leosattler@berkeley.com

March 7, 2014

Abstract

Through this lab we conduct several experiments as an introduction to the data analysis that radioastronomers perform. The following devices are used: two signal generators (also called local oscilators), providing sinusoidal waves at desired frequencies, a clock set at $200Mhz$ to perform the data aquisition, different types of mixers, a sampler (Pulsar) and the Roach, Reconfigurable Open Architecture for Computing Hardware, an open-source programable board. With them we manipulate different signals under different conditions and present the results. Toogether with the Fourier Transform we analyze data and prove the Nyquist Theorem, show the rationale behind Fourier Filtering, present the importance of frequency resolution and the working of heterodyne mixers, dealing with analog and digital mixing. We also use the Roach to perform many of these tasks introducing the capabilities of FIRs (Finite Impulse Response) filters, programing its circuits and understanding FPGAs (Field Programmable Gate Arrays), Logic Gates and other digital components.

1 Introduction

Digital cirucuits play an important roll in modern society, being part of a significant amount of everyday technologies, such as computers. Also, they are intrisically related to radioastronomy since are used in signal processing. Through a set of experiments we use the Nyquist criterion, Fourrier analysis and different *Python* libraries to understand the rationale behind digital devices, simulating with a signal generator what antennas in radioastronomy receive from the sky.

In Section 2 we use two Signal Generators and a sampler (Pulsar), setting them via the DEFEC (Digital Front End Control) *Python* module. With the *set_src* function we generate waves specifying the frequency, the amplitude (Vpp, voltage pick to pick), the power (in dBm), Voltage DC offset and the phase. With the *sampler* function we perform sample incoming signals with desired *number of samples* and *sampling frequency.* These last two receive great attention when defined. With this, we give the first steps on signal processing by dealing with Fast Fourier Trasforms (FFT's) and presenting the results of different signals being sampled at the same frequency.

In section 3 we work with heterodyne signal processing. Now sampling mixed waves, we understand the arithmetics of *DSB* (*Double Sideband*) and *SSB* (*Single Sideband*) mixers. We start to use the Roach by making similar sets of experiences and discussing the results.

In section 4 we use the Roach and perform severall data analysis, and understand the Down Conversion method, largelly implemented by radioastronomers on data aquisition.

2 First Samples

Incoming signals are analyzed in severall steps, and they are always performed in such a way that the less processing power will be required to generate the best (reliable, well resolved) data possible. It is not easy, and to understand how astronomers deal with it and the parameters behind data aquisition, we start presenting some graphs, the equations behind them and what their features mean.

2.1 Choosing Parameters

When setting the parameters for the Pulsar using the *sampler* function, the number of samples N_{sampl} is how many data points our digitally processed signal is going to show, while the sampling frequency ν_{sampl}

holds the rate that we are recording these points. For exemple, with a frequency $\nu_{sampl} = 20\ KHz$, we are sampling the incoming signal every $1/20\ [KHz]^{-1} = 50\ Microseconds$. If we set $N_{sampl} = 10,000$ points at this ν_{sampl}, it means that we would take 0.5 *seconds* performing this task.

One has to wisely choose these parameters, since they are related to the *Spectral Resolution* of our sampled data, defined as

$$\Delta f = \frac{\nu_{sampl}}{N_{sampl}}, \tag{1}$$

telling how well we can resolve the features of our spectrum profile. We note that smaller the Δf, better is our resoltution. Although a small sampling frequency or large amount of samples result in a small spectral resolution, they also will require more time for your sampling, as shown above. Also, this require more processing power, which means more physical resources.

A nice way to balance this equation is respecting another one, given by the Nyquist criterion. It tells that when we sample at 2 times the frequency of the incoming signal, we can recover a good wave profile that decently describes our original one. So, if we have

$$\nu_{sampling} = 2 \times \nu_{signal}, \tag{2}$$

we can sacrifice some data points and still recover a nice description of the input signal ν_{sig}. Fig.1 shows the result of a set of 9 different input signals being sampled by the same frequency $\nu_{sampl} = 10\ MHz$, using $N_{sampl} = 256$ data points.

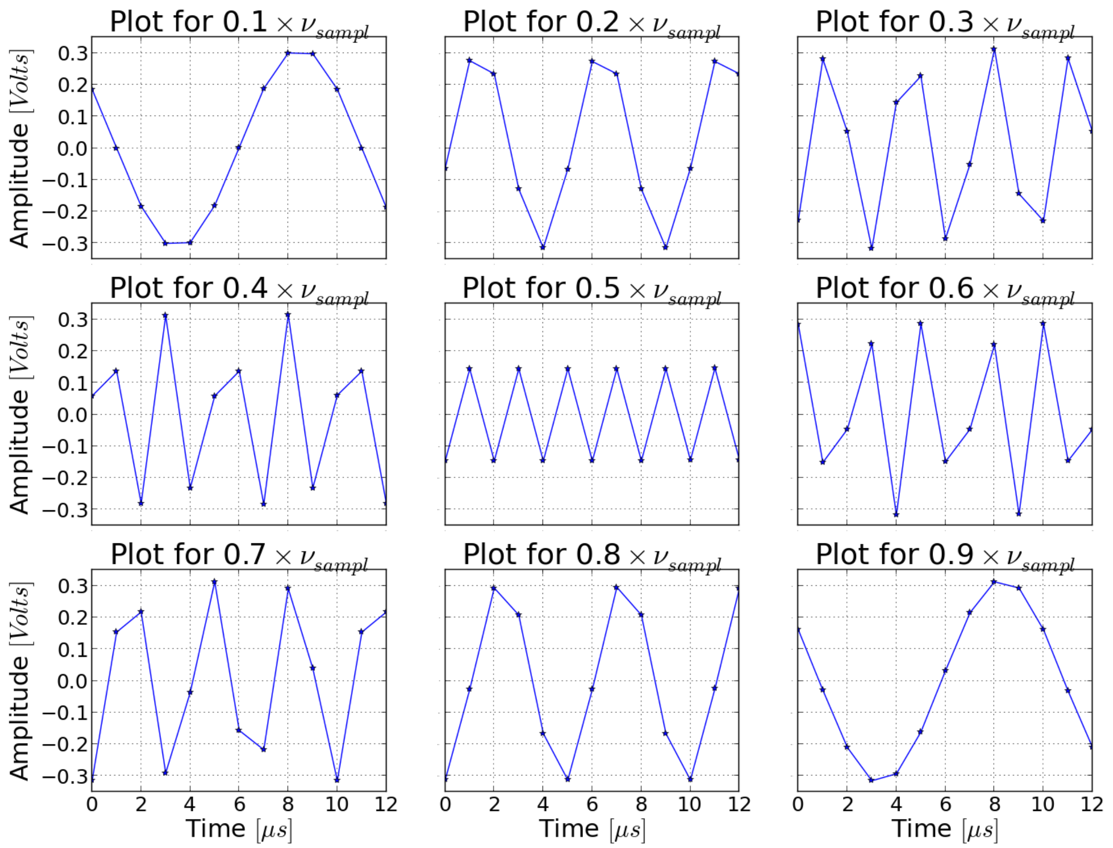

Figure 1: Plot for the sampling of 9 input signals, going from $\nu_{sig} = 0.1 \times \nu_{sampl}$ up to $\nu_{sig} = 0.9 \times \nu_{sampl}$, always with $N_{sampl} = 256$. The plots are zoomed in so that the wave features are best visualized.

Originally on the x axis we had the sampling numbers counting from 0 to 256, but one can easilly recover the physical (time) axis by multiplying the values by $1/\nu_{sampl} = 10^{-6}$ *seconds*.

It is easy to identify a similar pattern for the plots equally spaced from the middle one. This is because, changing the input frequency by multiples of ν_{sampl}, the sampling rate will be able to generate outputs that differ simply from a complete phase shift (or half of it, inverting the wave), and the data points (star dots in the graphs) will keep equally spaced, building the same profile after this shift. Also, this shift existis because, even by setting the sampler and the local oscilator with a *Python* code, puting each value inside a for loop that automatically (and instantly) changes it, there still exist a time span between each sampling performed.

In the case of the middle plot, since we have $\nu_{sampl} = 2 \times \nu_{sig}$, we are sampling under the nyquist criterion (see Eq.2). So this is the plot that best represents our original signal.

2.2 FFT, Spectral Leakage and Aliasing

If we want to analyze the signal on the frequency domain, the Fourier Transform ($\mathcal{F}$) says that

$$\mathcal{F}(\widehat{f}(\nu)) = f(t) = \int_{-\infty}^{\infty} \widehat{f}(\nu) e^{-2\pi i \nu x} d\nu, \tag{3}$$

the $\widehat{}$ denoting a Fourier-domain quantity and $\mathcal{F}^{-1}$ the inverse Fourier Transform. The next figure shows the different frequencies recovered when applying the *FFT* (Fast Fourier Transform) to the waves from Fig.1. Done by importing the *numpy.fft* module in *Python*, the FFT function is a way to efficiently calculate the discrete Fourier Transform (DFT) (discret version of the previous equation) by using symmetries in the calculated terms.

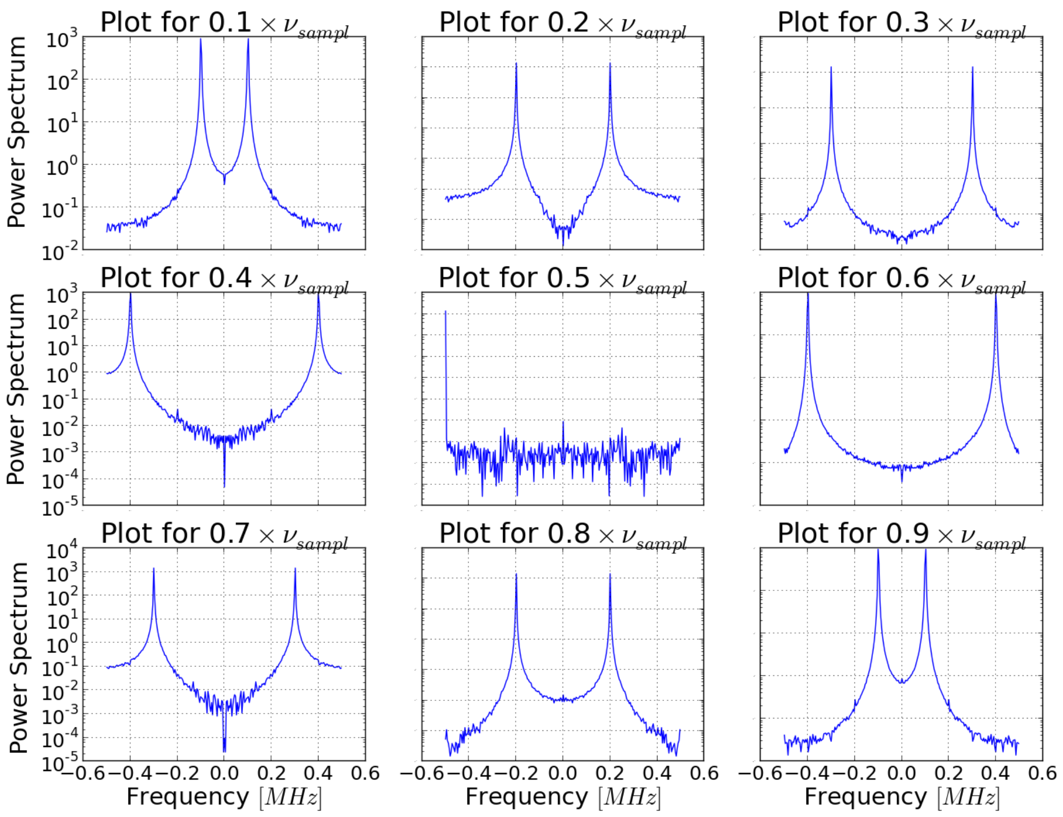

Figure 2: Fourier Transform applied to each sampled data showing their profiles on the frequency domain. The y axis are in *log* scale.

The purpose of Fig.2 is to illustrate the *Alisasing* phenomena. It occurs when the Nyquist criterion is not obeyed, so the frequency must deviate from its original value when sampled. As we can see, from the top left to bottom right, the frequency representation gets worse as ν_{sig} approaches ν_{sampl}, getting reasonable values on the fist four graphs and being totally *aliased* on the last four. At $\nu_{sig} = 0.5 \times \nu_{sampl}$, (one can also read at $\nu_{sampl} = 2 \times \nu_{sig}$), as expected the frequency representation is optimal and corresponds to the original value, having a well defined spike at 0.5 MHz.

We can also observe that in the middle plot there are no featured wings, present in all the other ones. This is called *Spectral Leakage*, also caused by aliasing but more intrisically related to the FFT opperation, which creates new frequency components around the sample bin, making the spikes look broad when not in the Nyquist frequency.

3 Mixing Waves

3.1 Analog Mixing

We now combine two different signals on a DBS (Double Sideband) mixer and sample the results. With these two incoming signals, one representing a *local oscilator* and another a generic signal defined as $\nu_{sig} = \nu_{lo} + \delta\nu$ and $\nu_{sig} = \nu_{lo} - \delta\nu$w, we will perform two data aquisition by first mixing the two waves and than sampling the result. Local oscilators ($LO's$) are used together with a mixer to convert any signal to another one with a frequency of interest. Being $\delta\nu = 0.5\%\nu_{lo}$, in one case the signal will be mixed with 1.05% (at -7 dBm) the value of $\nu_{lo} = 100$ kHz (at -3 dBm), and on the other with 0.95% (also at -7 dBm). Both signals are created and sampled with the same equipments of the previous section, and the result of an FFT applied to the outputs is shown in the next graph.

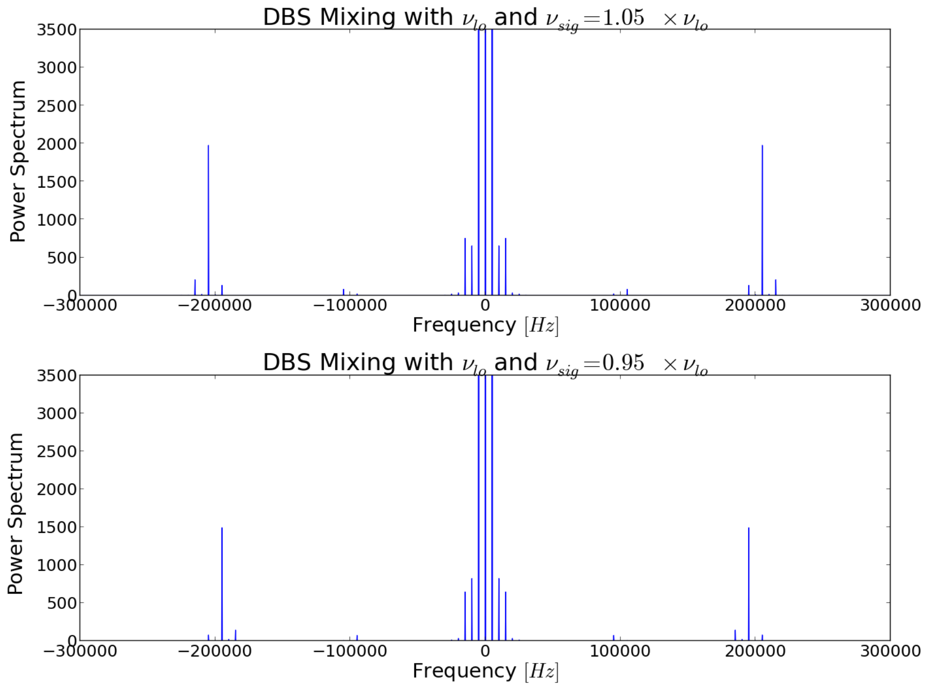

Figure 3: Graphs on the frequency domain presenting the output of a DSB filter. On the upper graph we have $\nu_{sig} = 105$ kHz and on the lower $\nu_{sig} = 95$ kHz, both cases mixed with $\nu_{lo} = 100$ kHz and sampled at 4 MHz (way above the Nyquist frequency). Y axis in Power Spectrum (no physical units).

The math behind this analog mixer is a sum and a subtraction of the LO and input signal frequencies. The output of the DSB is characterized in Fig.3, showing a simmetry and identical values of the upper and lower frequency output of the mixed signals. So for the two values of the incoming signal we would expect the presence of a frequency valued at 5 kHz, for the case when the mixer subtracts, and another around

200 kHz, when frequencies are added. Both outputs appear on both cases at Fig.3, where the lowest value is one of the spikes around zero and the highest can be easily spot. In each case we have a spike near the value without any wings, since the Nyquist criterion was well followed by sampling at 4 MHz.

The y axis is presented as a Power Spectrum, since indeed it is in units of Power. The signals that were complex after the FFT operation became absolute values by the *numpy.abs(f)* (f any generic argument, in this case the output of FFT). To get the values presented on the graph it was squared, $numpy.abs(f)^2$, hence it has units of $Volts^2$, but there's no associated resistance value to give a physical power. Besides that, these volts are sampled bits, also faked by many of the intrinsic voltage values (noises, DC offsets) related to the devices, so this output gives no solid information about the Power of the original incoming signal.

3.1.1 Fourier Filtering

In this section we take a look at this same output but in time domain. On the first case, where we mix $\nu_{sig} = 105\ kHz$ and $\nu_{lo} = 100\ kHz$, the DBS mixer produces high harmonics that end up as spikes over the main frequency, due to the sum operation that this mixer realizes as described on the previous subsection. One can get rid of these spikes by dealing with the Fourier space and aiming a frequency to be eliminated, in this case the high ones. The result is on the next graph, where the top plot contains the output of the mixed frequencies, and the bottom one is the result of a Fourier filter. The idea behind this method is to make usage of the Fourier space: by looking at the top plot of Fig.3 it is easy to determine which frequencies we want to eliminate, so on frequency domain (after applied a FFT to our data) we can set to zero the correct data points related to these high values. Once done this (intrisically a sort of filtering), we can recover the time domain by using the Inverse Fast Fourier Transform from the *numpy.fft* module, called *numpy.fft.ifft*. This is a mathematical filter but, as shown on Fig.4, decently recovers the signal as desired.

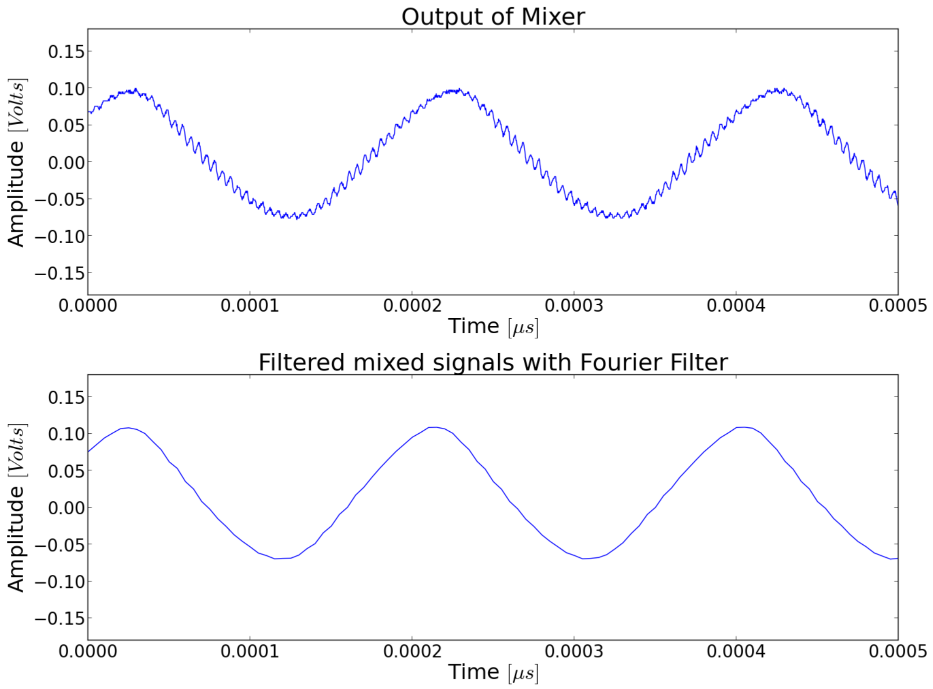

Figure 4: Fourier Filter applied to the output of mixed signals. Once specified the frequency we want to filter, we apply the FFT to mathematically manipulate our data, and then use the FFT^{-1} to recreate our signal now without the aimed frequencies. The graph was zoomed in so that the spikes could be visualized, and the time scale was ideally recovered by setting the x axis to be the inverse of the sampling frequency times the number of data points.

3.2 Digital Mixing

We now advance on the world of data analysis with digital devices by dealing with the Roach and its default programs. They are the bridge between us (humans) and the machine when we want to digitally represent our analog signals. To do so we access the Roach interface by typing *ssh root@roach* on the *Linux* terminal shell.

First we perform a digital DSB mixing, by sending the same frequencies of the previous subsection to the Roach. We input the signals on the *ADC card*, which has 6 ports (the third one being used only for the clock). For this experiment our signals went to ports 2 and 3, and we started by lauching a *BOF* (.bof) file that interacts with the Roach. This is in fact a program to be written on the Roach's FPGAs and will command a set of tasks, in this case will be mixing and sampling our input signals. The samples are read on memories named as, for example, *adc_bram, mix_bram, sin_bram* and *cos_bram*, where *BRAM* stands for *block random access memory*. A register called *trig* will allow the samples to be written on the memory files by setting its value to 1 (to start writing) and 0 (to stop). This is done by typing $echo\ -ne$ $\backslash x00\backslash x00\backslash x00\backslash x01$ to initialize the data taking. This is a hexadecimal 1, and that is how we make comunications with the digital world. We type $\backslash x00\backslash x00\backslash x00\backslash x00$ at the end of the same command, now reading zero, to stop the data aquisition.

This is the procedure to be followed in order to write data over one of the memory files presented above, giving you the digitally manipulated version of your analog signal acording to your inputs and to the .bof file that you set to launch. We ran the *adc_snaps_2014_Feb_13_2111.bof* file and wrote inside *mix_bram* two input signals, identicals to the ones used with the DBS analog mixer ($\nu_{sig} = 105\ kHz$ and $\nu_{lo} = 100\ kHz$). The result is presented in the botton of the following figure, while the top plot is the same as the top one of Fig.3, both graphs in frequency domain. Next, we see the output in time domain of this mixer (Fig.6), with an incoming wave of 1 MHz mixed with another of 1.5 MHz.

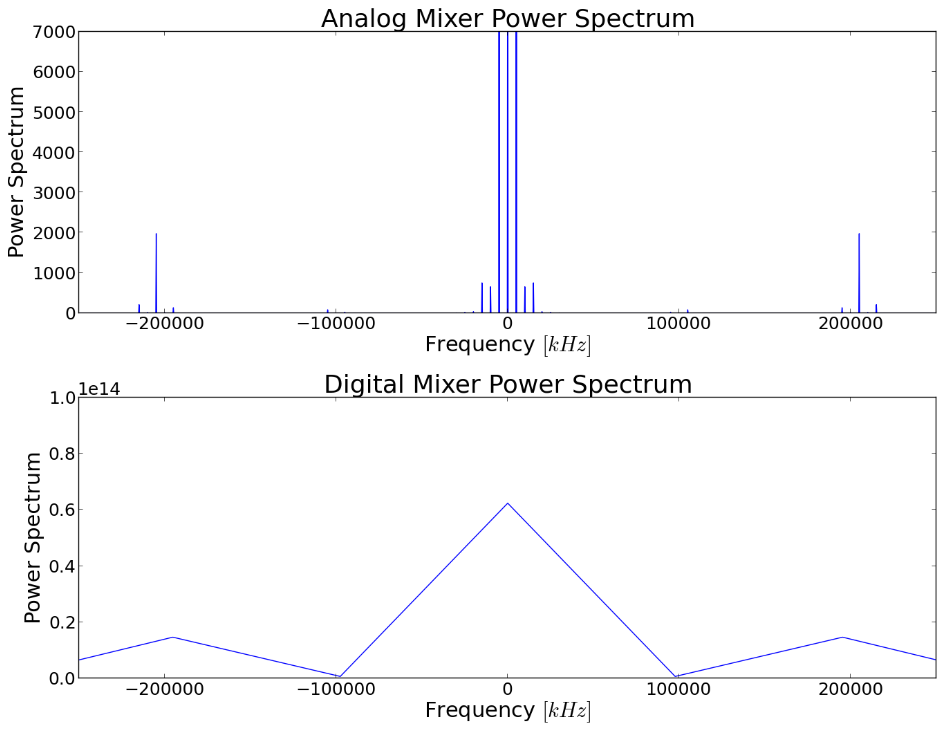

Figure 5: Plot showing the Power Spectrum of the outputs of the Analog DBS Mixer (top) and Digital DSB Mixer (bottom). The Power Spectrum comes as the label for the Y axis once again (see Fig.3). The high values for the bottom plot is discussed below.

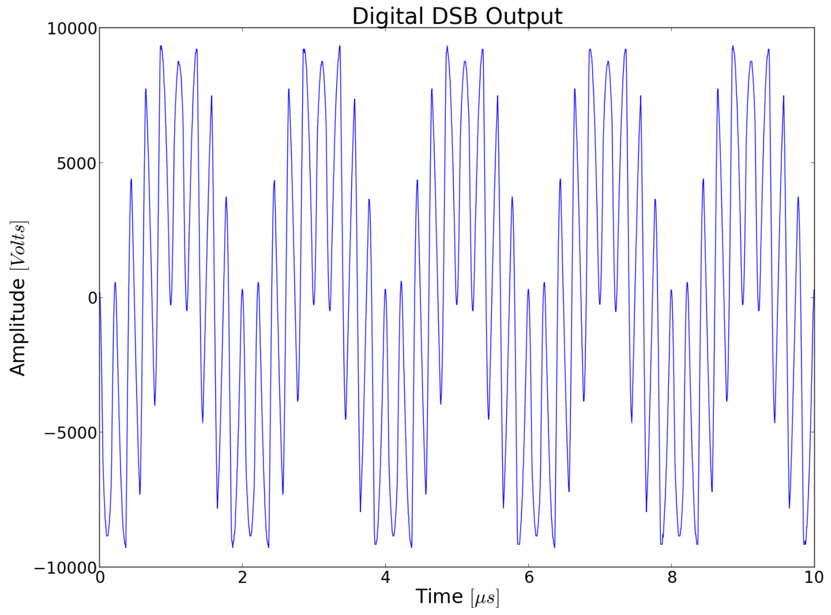

Figure 6: Plot of the output from the digital DSB mixer. The x axis is in microseconds since this is the time domain.

The range of the y axis of the plots on Fig.5 seems not to agree with the value of the amplitude, in $Volts^2$, of the incoming signals. If we could guess, it would be reasonable to assume the top plot as the correct one, since the power seems too high for the output of the digital mixer. This is because, in this case, we are applying the Fourier Transformation (FFT) to sine waves, which results in *Delta Functions* that have infinitely high values at a particular point and zero at all the others. So, when we have a (roughly) well defined frequency to be sampled, the FFT interpret as something similar to a sine wave and outputs high values.

Also, the bottom plot seems to represent the same data of the top one but in a poor way, and the reason is the spectral resolution related to the Roach procedure. According to Eq.1, the frequency resolution from sampling with the Pulsar is $\Delta f = 4 \times 10^6/16000 = 250Hz$, while with the Roach it is $\Delta f = 200 \times 10^6/2048 \approx 9.8 \times 10^4$. Basically, the frequency resolution of the Pulsar is 400 times better then the Roach's resolution, hece the better representation of the frequencies on the top plot.

3.2.1 SSB Mixer

A *SSB* or *Single Side Band* mixer works as a filter by completelly eliminating a band of the incoming mixed signals, generating outputs as the ones of Fig.7. In order to take these data, we set a clock at 200 MHz and an incoming signal of $\nu_{sig} = 100\ kHz$ to the Roach port number 4. The frequency of the local oscilator is defined by setting a value to the register *lo_freq* after running the same .bof file of the previous section, and then setting *trig* to 0 and 1, respectively.

The way we dialog with the Roach interface on this part tells a lot about how SSB mixers works. We first set a value to *lo_freq*, in this case 2, and together with the clock that samples at $\nu_{clock} = 200\ MHz$ the incoming signal, at each resulting sample a *sine* and a *cosine* wave will be evaluated with the frequency equals to *lo_freq*, and the result will multiply the incoming ν_{sig}. The memory files analyzed here will be *sin_bram* and *cos_bram*, containing the result of the input signal mixed with the sine and cosine calculated, respectively. At the end, the local oscilator frequency ν_{lo} for this mixer will be $\nu_{clock} \times lo\_freq/(256)$, where the value 256 is because this is the number of bits (i.e., sample size) of the clock sampling.

To reconstruct the data, we set the sin values to be the imaginary part of a sinusoidal wave, and summed with the cosine values we apply the FFT to build the graph of Fig.7. It shows the mixing of $\nu_{sig} = 100\ kHz$ and $\nu_{lo} = 1.56\ MHz$, as a result of setting *lo_freq*$= 2$. We would expect a sum and subtraction of frequencies, but Fig.7 only shows spikes around positive values. This is because SSBs can be carachterized either as

USBs (Upper Sidebands) or LSBs (Lower Sidebands), and in this case we have an USB, so only the result $\nu_{sig} + \nu_{lo} \approx 1.66\ MHz$ (above zero) is appearing.

However, as we can observe the spike is not exactly at $1.66\ MHz$, and this has a reason. Since we are sampling with $\nu_{clock} = 200\ MHz$ an amount of 2048 data points from the incoming signal, our frequency resolutions is $\Delta f \approx 98\ kHz$ (see Eq.1). This is practically the value of our ν_{sig} ($= 100\ kHz$), making it impossible to be well resolved and hard for the SSB mixer to work with reliable values from the sampled wave. In fact, it will be mixing values that roughly represent ν_{sig}. However, the spike is indeed located somewhere near $\nu_{sig} + \nu_{lo} \approx 1.66\ MHz$, as expected from the SSB mixer, and this is an acceptable description of how this mixer works. Besides, we also see a broad wing at zero. The reason being that the DC offset of the ADC card, connected to the 8-bit sampler of the Roach, has a small value that oscilates around zero and gives this noticeable feature.

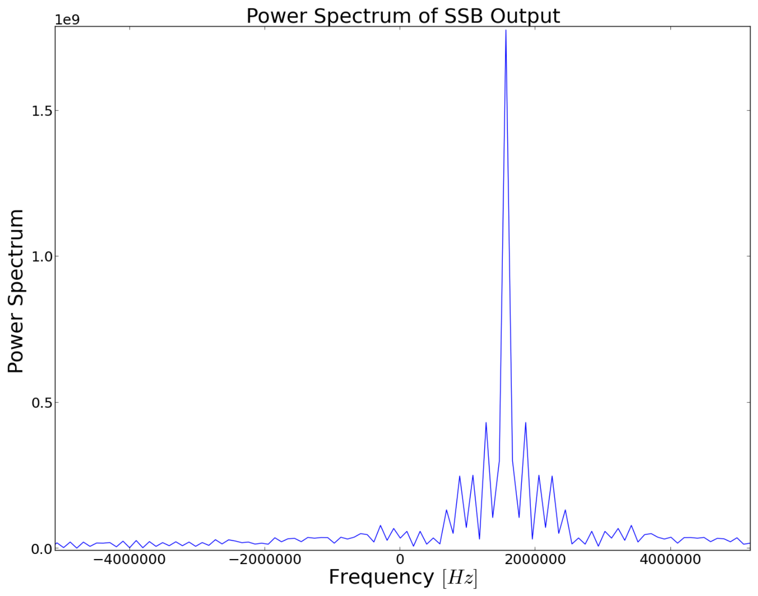

Figure 7: Output of the manipulation of *sin_bram* and *cos_bram*, showing that the SSB mixer is acting as a filter for lower frequencies (USB).

4 Digital Down Converters

We have been programing many tasks on the Roach that are important in signal processing. Now we deal even more with its interface, writing a large amount of information in register files and understanding *FIRs* (Finite Impulse Response) filters, in order to make usage of a *Digital Down-Converter*.

4.1 FIRs

4.1.1 Convolution

Being mostly implemented with FPGAs, Digital Down-Converters are largely used in Radioastronomy, and its main function is to mix and filter a signal. For so we create an FIR filter. They are digital circuits used to filter an input signal in frequency-domain by working with pre-set coefficients, that once determined will *convolve* in time-domain with an input waveform.

The *Convolution Theorem* states that a multiplication in frequency domain is the same as a convolution, defined as

$$[f * g](\tau) \equiv \int f(t)g(\tau - t)dt = \frac{1}{2\pi}\int \widehat{f}(\omega)\widehat{g}(\omega)e^{i\omega\tau}d\omega, \tag{4}$$

in the time domain. To prove, we can rename $\tau \to t$, and from Eq.3, regarding that the $\widehat{}$ represents Fourrier-domain function, we have

$$f(t) * g(t) = \int \widehat{f}(\omega)\widehat{g}e^{i\omega t}d\omega = \mathcal{F}^{-1}(\mathcal{F}(f)\mathcal{F}(g)), \tag{5}$$

the last product being between two frequency domain functions, since we have an inverse Fourier Transform ($\mathcal{F}^{-1}$) applied to them.

This operation is the heart of our FIR filter. Samples from an incoming signal in time domain will be multiplied by time-like coefficients and the result will be summed along, as more and more samples are being computed. This task, a convolution, is performed inside the FPGA of the Roach after we run the *dig_dwn_conv_2_2014_Feb_25_1332.bof* file, choose an appropriate *lo_freq* for a Local Oscilator, set the coefficients and trigger the capture. However, choosing values to be written on the coefficient registers requires deeper analysis of how the FIR works.

4.1.2 Choosing Coefficients

As said, the FPGA inside the Roach will receive a large amount of tasks and information to implement our Digital Down-Converter, and it includes the coefficients for the FIR filter. They come from the following rationale: the product of two functions, one describing random frequency features of a signal and the other being a square wave, will result in a profile that seems filtered, since the square function tends to zero out what is outside its non-zero range of values. The first could be an incoming wave analyzed in frequency domain, and the second could be represented by an array of zeros and consecutive ones. Multiply both and the incoming wave is filtered. However, our signal comes in time domain, and our square wave is nothing but an ideal filter, inexistent.

But product of frequency-like functions is a convolution for time-like ones (as just seen above), and they are apart from each other only by a Fourrier Transform (in this case, inverse)! So the coefficients are given by the inverse FFT of this array of zeros and ones with 8 elements, and the result is presented on Fig.8.

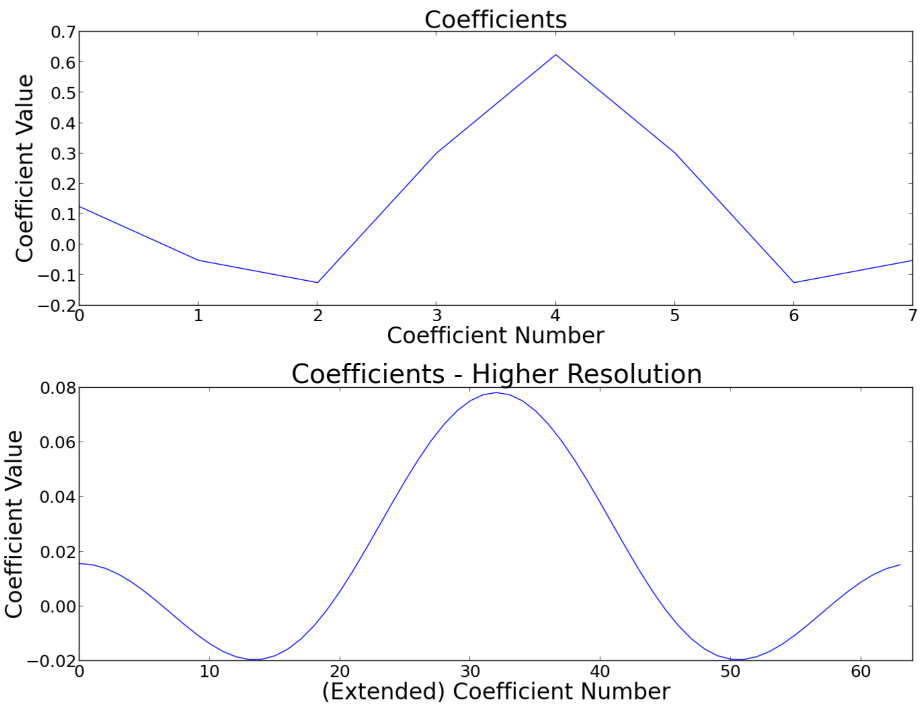

Figure 8: *Top*: Plot showing the calculated coefficient values and the filter response. *Down*: Result of applying more points (zeros) to the filter bins, enlarging the resolution.

This was performed using the function *numpy.fft.ifft*, and the output is an array whose values are Complex numbers. But its imaginary parts are negligible, so Fig.8 has only the real values of the FFT^{-1} applied to our array. The bottom part is the filter in a higher frequency resolution, after added zeros to the original array in order to have 64 samples.

But our coefficients, just as the *trig* files, must be $32-bit$ signed integers with 17 bits after the binary point in order to be interpreted by the FPGA. More than that, must be in hexadecimal representation. Since we have decimal floating point numbers, the first step is to convert them to binary. It was done with a developed *Python* program (called *dec_bin.py*), which can be found at my github page (here). It converts any positive decimal number, smaller than 1 and greater than 0, to a binary number with 18 values, being 17 after the binary point. Since some of our coefficients were negative, this program was used to calculate its positive value and the rules of the *2's complement* representation were used to have them as negative.

With the 18 bits numbers, because they are the ones that matter to our $32-bit$ value, we can set the fisrt 14 to be zero and write them into hexadecimal representation. Table 1 was build in order to help this procedure and the result is the third column on Table 2. The first column of table 2 shows the calculated coefficients, and the second has the output of the *Python* program, with extra calculation for negative coefficients. To get the hex representation, we devided the Binary values in groups of 4 and, using Table 1 as reference, wrote from righ to left the correspondent value in hex. After each $\backslash x$ we are supposed to write 1 byte (8 bits, or 2 hex values), and as a whole our hex number holds the 32 bis to be written as the coeeficient for our FIR filter.

Converting Table		
Decimal Value	Hex Value	Binary Value
0	0	0000
1	1	0001
2	2	0010
3	3	0011
⋮	⋮	⋮
9	9	1001
10	a	1010
11	b	1011
12	c	1100
13	d	1101
14	e	1110
15	f	1111

Table 1: Table used as reference to convert the binary points into hexadecimal values.

Coefficients		
Decimal Floating	Binary Fixed Point	Hex representation
0.125	0.00100000000000000	$\backslash x00\backslash x00\backslash x40\backslash x00$
-0.0517767	1.11110010101111101	$\backslash x00\backslash x03\backslash xe5\backslash x7d$
-0.125	1.11100000000000000	$\backslash x00\backslash x03\backslash xc0\backslash x00$
0.3017767	0.01001101010000010	$\backslash x00\backslash x00\backslash x9a\backslash x83$
0.625	0.10100000000000000	$\backslash x00\backslash x01\backslash x40\backslash x00$
0.3017767	0.01001101010000010	$\backslash x00\backslash x00\backslash x9a\backslash x83$
-0.125	1.11100000000000000	$\backslash x00\backslash x03\backslash xc0\backslash x00$
-0.0517767	1.11110010101111101	$\backslash x00\backslash x03\backslash xe5\backslash x7d$

Table 2: Coefficients used for the Digital Down Conversion on the Roach. Each number after the backslash (\) and x represent the value of 4 bits, so each two numbers holds a byte. We can see that summed they evaluate the 32 bits for the value of *coeff*.

4.2 Using Coefficients

To implement our coefficients, a noise source, with a frequency range of 200 MHz, was attached to the Port 1 of the Roach, and after launching the correct *.bof* file and setting *lo_freq=2*, the result is presented in Fig.9.

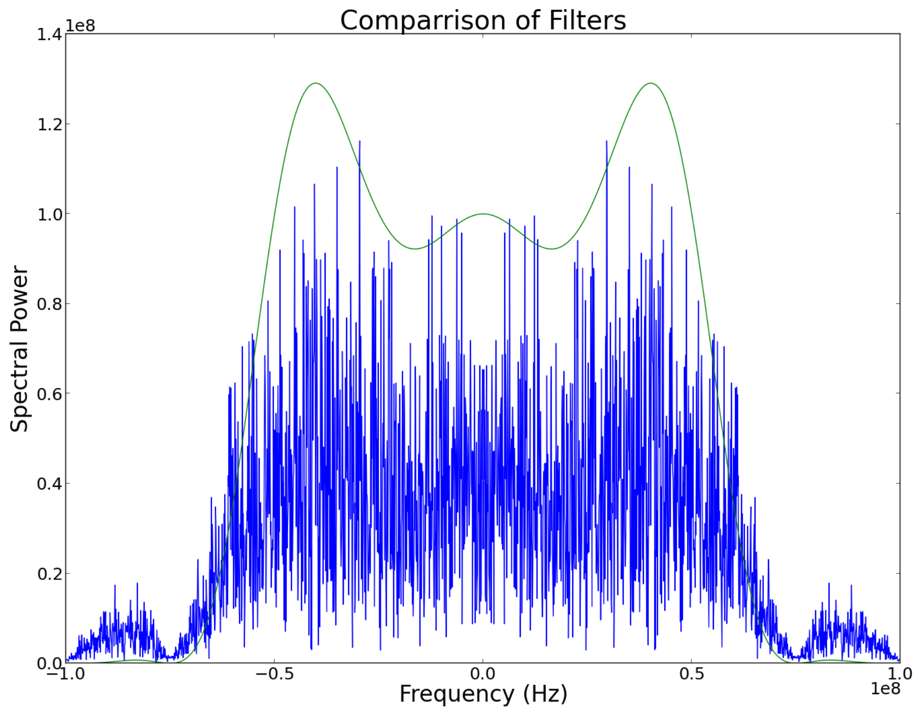

Figure 9: Result of the implementation of the coefficients presented on Table 2. The green line is the theoretical filter shape, and the blue is the output of the filter response.

As expected, they have the same shape, but the theoretical values had to be rescaled since the output of the filter has high values and should be normalized by the value of the noises without being filtered.

5 Conclusion

In this lab we explored the relation between sampling frequency and frequency resolution, a subject of great importance in data analysis, and proved the Nyquist Theorem showing how it affects our signal representation, as observed by the analysis of the information on Fig.2. The Fourier Transform, just as the *numpy.fft* functions, were largelly explored, showing its relevance in signal processing. *Python* codes were devolped to its implementation. As an example, Fig.4 shows the manipulation in Fourier space of a mixed wave by cancelling desired frequency values, known as Fourier Filter.

The characteristics of Heterodyne mixers were explored, their difference high-lighted. For the DSB mixer, we presented the difference in frequency resolution, defined by Eq.1, between the analog and digital ones, and how it affects the Power Spectrum of the mixing of two waves. Also, we showed the working of SSB mixers and the steps performed to acquire its data from the Roach.

We had an important introduction to the digital world, developing skills to understand the way to write hexadecimal values into the interface of the Roach, dealing with binary numbers and the *2's complement* representation. By understanding the Convolution Theorem and the architecture of Digital Down Converters, we presented the steps required for the implementation of an FIR filter, and compared the result of our *theoretical* filter shape, from theoretically calculated coefficients, with the shape resulting from its operation over a noise source. As expected, the shapes were equivalent.