Discrete time real signal processing, introduction using GNU Radio Companion

J.-M Friedt, November 7, 2018

Many radiofrequency signals acquired using basic hardware setups are real (as opposed to complex) signals. The property that the Fourier transform of real signals is even make their processing challenging, with risks of injecting in the band being analyzed around baseband some unwanted spectral components. We will analyze a classical processing step – the Hilbert transform – which, based on a real signal, generates an imaginary component designed to eliminate the negative part of the spectrum and hence ease the processing of the acquired samples.

1 Introduction

A radiofrequency signal has been transposed in frequency, by the emitter, in order to allow sharing the electromagnetic spectrum resource amongst multiple users, and possibly to efficiently emit the signal using an antenna whose dimensions are compatible with the object in which the emitter is embedded – as a reminder, the characteristic dimensions of an antenna is the half wavelength $\lambda/2$, with $\lambda = 300/f$ when the signal is emitted at f MHz. Upon reception, the signal must be moved back from its radiofrequency band – around the carrier frequency – to baseband centered on 0 Hz to be processed (Fig. 1). A simple way of frequency transposition is to multiply in the time domain: a signal $s(t) = \exp(j\omega t)$ is brought within the analysis bandwidth of the receiver sampling at f_s if mixing with a local oscillator of angular frequency ω_{LO} following $s(t) \cdot \exp(-j\omega_{LO}t) = \exp(j(\omega - \omega_{LO})t)$ meets the condition $|\omega - \omega_{LO}| \leq \pi \cdot f_s$. Once this frequency transposition has been achieved, the analog to digital conversion provides a datastream that can be once more frequency transposed, digitally this time, using a new multiplication with $\exp(j\omega'_{LO}t)$ using this time a discretized time t synthesized with $1/f_s$ steps, or in Matlab syntax `t=[0:length(s)-1]/fs;`

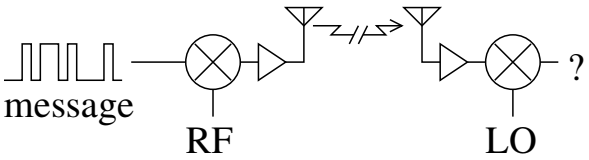

Figure 1: Transmitting a `message` over a radiofrequency carrier aims at sharing the spectrum between multiple speakers and make the antenna dimensions compatible with the envisioned application. However, the frequency of the receiver local oscillator LO is not synchronozied with that of the emitter: various strategies for locking the receiver on the emitter are needed (demodulation) in order to decode the message on the receiver side.

So far we have considered an ideal case, in which signals are complex, make the spectral analysis easier since $\exp(j\omega t)$ only exhibits a single spectral component at ω. However, an antenna collects a voltage, which is a real signal, and a radiofrequency synthesizer generates a voltage, a real valued quantity as well. A real signal exhibits two spectral components at $+\omega$ and $-\omega$ since $\cos(\omega t) = \frac{1}{2}(\exp(j\omega t) + \exp(-j\omega t))$. Even without going through a mixer for frequency transposition [1], connecting an antenna to a sound card for example allows for sampling a voltage, again a real quantity. The concept of an imaginary part is generated by introducing a second component in quadrature, following the notation $\exp(jx) = \cos(x) + j\sin(x)$ which indeed exhibits a real part (cos) and an imaginary part (sin) in quadrature ($\cos(x) = \sin(x + \pi/2)$). Many receivers only measure a voltage, a not the two quadrature components as provided by an I/Q detector. We have mentioned using a sound card as a Very Low Frequency (VLF [1]) receiver, but such a condition is also met in case of seismic measurements or RADAR measurements (aerial or ground penetrating [2]), or any receiver using a single mixer for frequency transposition, or actually directly sampling the radiofrequency band.

The issue with sampling a real-valued signal is that the imaginary part of the spectrum exhibits a magnitude equal to that of the real part (Fig. 2). All the spectral components visible in the positive frequency part of the spectrum are found on the negative part as well. Such an even spectrum becomes an issue when the digital signal processing on the real datastream aims at shifting the frequency – using digital multiplication of the signal – towards a lower frequency band. Indeed, during this operation, the whole spectrum is shifted along the abscissa axis, due to the periodicity of the spectrum of the discrete-

[1] multiplying in the time domain yields an addition in the frequency domain, since $\exp(j\omega_1 t) \times \exp(j\omega_2 t) = \exp(j(\omega_1 + \omega_2)t)$

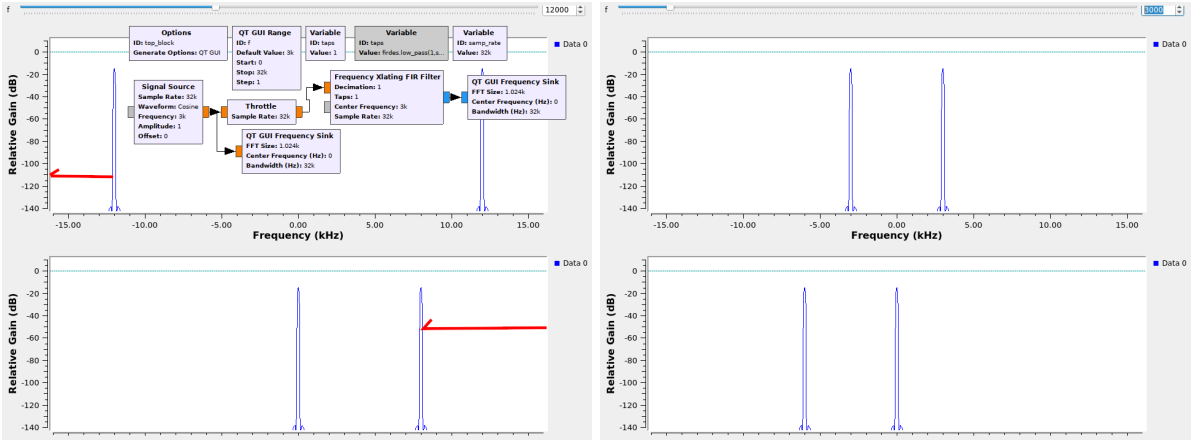

Figure 2: Impact of frequency transposition of a real valued signal: right, transposing a 3 kHz signal sampled at 32 kHz keeps the negative component of the signal in the [-16:0] kHz range, no aliasing is observed and filtering the negative frequency spectral component is easy. Left: a 12 kHz signal, meeting the sampling theorem, exhibits a negative frequency spectral component shifted to the positive frequency band due to the periodicity hypothesis of the spectrum of the discrete-time signal, since $2 \times (-12)$ kHz$< -32/2$ kHz. This positive frequency spectral component might reach the 0-frequency or at least the baseband, introducing noise during the baseband signal processing.

time signal: any signal reaching below $-f_s/2$ is brought to the other side of the spectrum at $+f_s/2$ and reciprocally: this effect is named aliasing, which can be used efficiently when the spectral components are well understood and controlled, but which will he be considered as a nuisance.

2 Impact of the Hilbert transform in the frequency domain

We have seen that a periodic real signal expressed as $\cos(\omega t)$ can be written as the sum of two complex exponentials $\exp(j\omega t) = \cos(\omega t) + j\sin(\omega t)$ as can be easily demonstrated by remembering that $sin(-\omega t) = -\sin(\omega t)$: $\cos(\omega t) = \frac{1}{2}(\exp(j\omega t) + \exp(-j\omega t))$, demonstrating that indeed a real periodic signal exhibits two spectral components located at $+\omega$ and $-\omega$. Furthermore, the expression of $\exp(j\omega t)$ which only exhibits a single spectral component at $+\omega$ leads us towards the path to be followed, at a single given frequency, to eliminate the negative spectral component: add an imaginary component to the signal in quadrature to the real part. Due to the linearity of the Fourier transform, any signal can be decomposed as a sum of various spectral components, and this operation can be performed on each one of the spectral components, hence generalizing the concept to any real signal acquired. This mathematical operation aiming at adding an imaginary part designed to cancel the negative frequency spectral components is called the **Hilbert transform**.

Reading the source code of the implementation of `hilbert.m` in the Signal Processing Toolbox of GNU/Octave shows that practically, no identification of the quadrature components of the initial signal are identified. The Fourier transform of the real valued signal is computed, its negative frequency components defined as null, and the inverse Fourier transform of the resulting dataset is computed, hence generating by definition an imaginary component in quadrature with the real component as shown earlier: `f=fft(f);f=[f(1,:); 2*f(2:(N+1)/2,:); zeros((N-1)/2,W)]; f=ifft(f);`. On the opposite, GNU Radio uses in `github.com/gnuradio/gnuradio/blob/master/gr-filter/lib/hilbert_fc_impl.cc` an implementation of the Hilbert transform as a Finite Impulse Response (FIR) filter following the methodology described for example in [3], since the coefficients of the filter are defined by `firdes::hilbert(d_ntaps, window, beta);`. We indeed find in `github.com/gnuradio/gnuradio/blob/master/gr-filter/lib/firdes.cc` the filter used to approximate the impulse response $h(t) \propto 1/t$ since `float x = 1/(float)i;` with `i` the index of each element in the filter. The details of the band pass filter allowing to get rid of the unwanted impact of the finite bandwidth of the filter associated with the number of coefficients are described in details in [4, section3.37, pp.168–177].

These concepts are investigated using GNU Radio (Fig. 3) by analyzing the spectrum of the signal

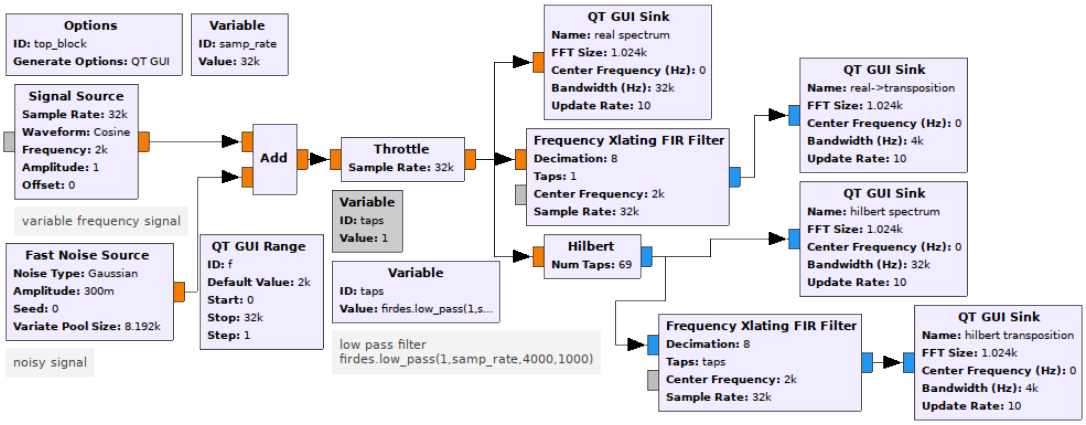

Figure 3: Impact of the Hilbert transform on a signal which is frequency transposed and decimated, inducing with the second operation a narrower analysis bandwidth and increasing the risks of aliasing.

before and after frequency transposition and decimation in order to zoom in the frequency range of interest around baseband and get rid of the high frequency components of the spectrum which are no longer of interest, and this with or without Hilbert transform (Fig. 4). We observe that the parasitic spectral component has been eliminated in the latter case.

3 Impact of the Hilbert transform in the time domain

The Hilbert transform is used during RADAR signal processing for extracting the envelope by computing its magnitude (Fig. 5). Indeed, while the signal $A(t)\cdot\cos(\omega t)$ exhibits a periodic components with angular frequency ω, adding the imaginary part $jA(t)\cdot\sin(\omega t)$ produces $A(t)\cdot(\cos(\omega t)+j\sin(\omega t)) = A(t)\exp(j\omega t)$ whos modulus is $|A(t)|$ since $|\exp(j\omega t)| = 1\ \forall t$. Similarly, the phase of the signal thus computed represents the delay of the signal.

As in most cases when performing digital signal processing, *removing the mean value of the real signal* is mandatory prior to applying the Hilbert transform.

4 Transposition and décimation, and the need for filtering

We have just discussed decimating the data flow. In a radiofrequency digital signal processing chain, the amount of information can only decrease but never increase by being created along the chain. We start by sampling the signal over a wide bandwidth including all the spectral components of interest, and then demodulate to extract the information content, and finally analyze the message payload at baseband to generate letters (bits), words (bytes), and sentences (checksum and error correcting codes) ... each of these steps requiring a narrower bandwidth.

The spectrum of a signal sampled at f_s spans from $-f_s/2$ to $+f_s/2$. On the other hand, the processor must process data at a rate of f_s samples/second. If f_s is large, only basic operations can be applied to the datastream without risking loosing samples, due to the finite computational processing power. However, shortly after the first processing steps (frequency transposition and filtering, as will be seen below), the spectrum has become too wide and part of its spectral components are no longer needed. How can we keep only part of the spectrum ? In the time domain, by only keeping one in N samples. Doing so – decimating the datastream – we reduce the datarate and hence the number of samples to be processed every second: for a given computational power, we can address more complex operations (e.g. FM demodulation, Costas loop). The counterpart in the frequency domain is to consider a spectrum spanning from $-f_s/(2N)$ to $+f_s/(2N)$: we only focus on the central part of the spectrum and get rid of the parts ranging from $f_s/(2N)$ to $f_s/2$.

However, what happens to the spectral components included in the band ranging from $f_s/(2N)$ to $f_s/2$ during the decimation step ? It is brought to baseband $[-f_s/(2N), +f_s/(2N)]$ due to the aliasing effect. If we do not want to benefit from this effect on purpose, we must remember to *filter* out the initial signal prior to decimating, with a filter cutoff frequency around $f_s/(2N)$ (Fig. 6). Since this filter is centered on

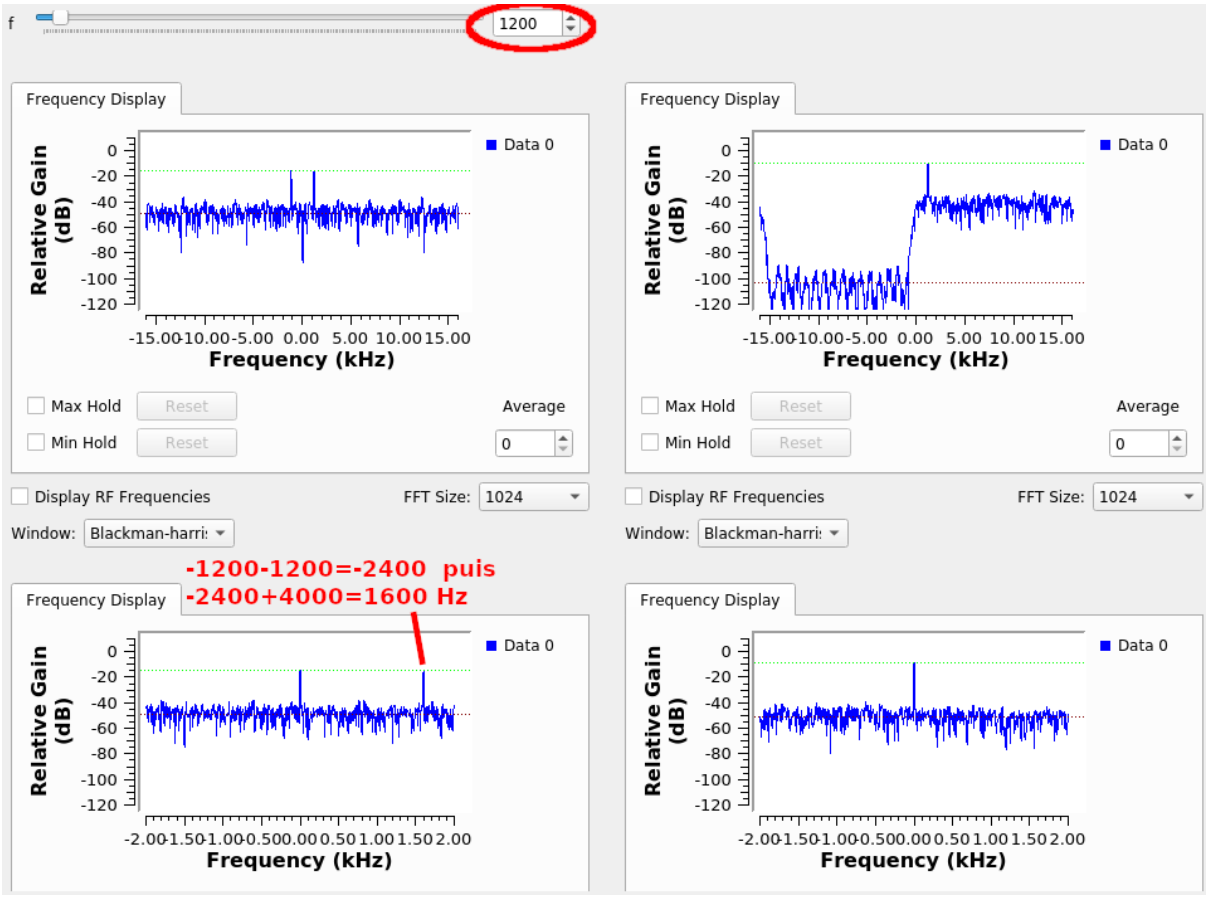

Figure 4: Top-left the real signal, and bottom-left the signal after transposition and decimation, inducing an unwanted spectral component in the righmost part of the spectrum. Indeed, the spectral component at -1200 Hz (frequency of the generated signal) is transposed to -2400 Hz by the Xlating FIR Filter, and then the decimation by a factor of 8 reduces the frequency range from -2000 to 2000 Hz, bringing the -2400 Hz spectral component to +1600 Hz by adding an integer number of sampling rates (spectrum periodicity assumption inducing aliasing). Top-right: the Hilbert transform was applied to the real signal, eliminating the negative frequency spectral component, which is hence no longer observed in the investigated band following the transposition and decimation (bottom-right).

0 Hz, it is a low-pass filter, which will automatically cut negative frequency components below $-f_s/(2N)$. Using GNU/Octave, decimation is achieved by keeping one in N samples using the notation `x(1:N:end)`, while filtering is achieved by applying the weighting coefficients `b` of a low pass filter (`filter(b,1,x);`) identified as `b=firls(M,[0 fs/2/N fs/2/N*1.1 fs/2]*2/fs,[1 1 0 0]);`. We leave as an exercise to the reader the detailed understanding of this filter definition which considers a transition width of about 10% of the sampling frequency, with a number M of coefficients selected to be around a few tens. The spectral resolution of a Fourier transform computed on M samples being f_s/M, we must select M large enough to be able to resolve the transition from $f_s/(2M)$ to $f_s/(2M) \times 1.1$ while keeping the computational load reasonable. The result is the elimination of the spectral component aliased during decimation, which might otherwise have crept into baseband if we had not been careful (Fig. 7).

5 From simulation to real signals

5.1 PlutoSDR for listening to 8 broadcast FM stations

We conclude this overview of GNU Radio Companion and discrete time signal processing by replacing the synthetic signal source with “real” signals. Rather than again using a DVB-T receiver used as general

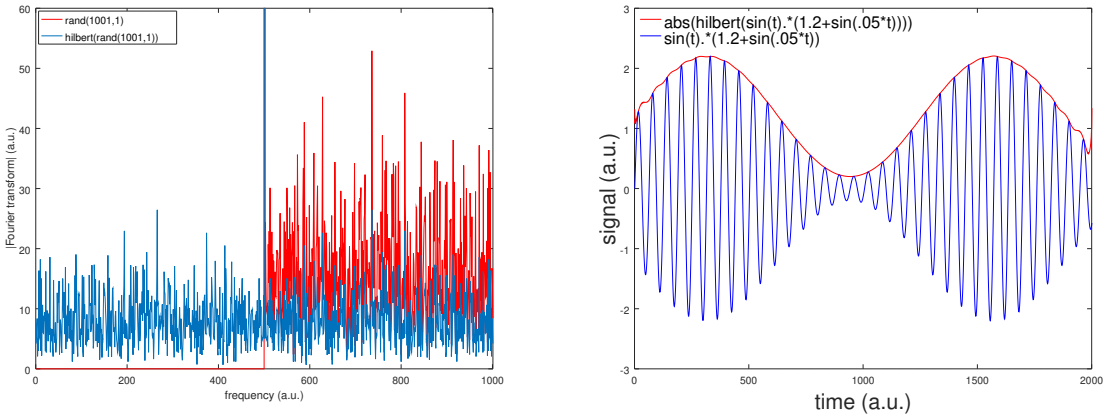

Figure 5: Using GNU/Octave, the `hilbert.m` fonction performs the same operation as depicted previously, eliminating the negative frequency component of the spectrum of the signal (left) or allowing for the extraction of the envelope by removing the carrier (right).

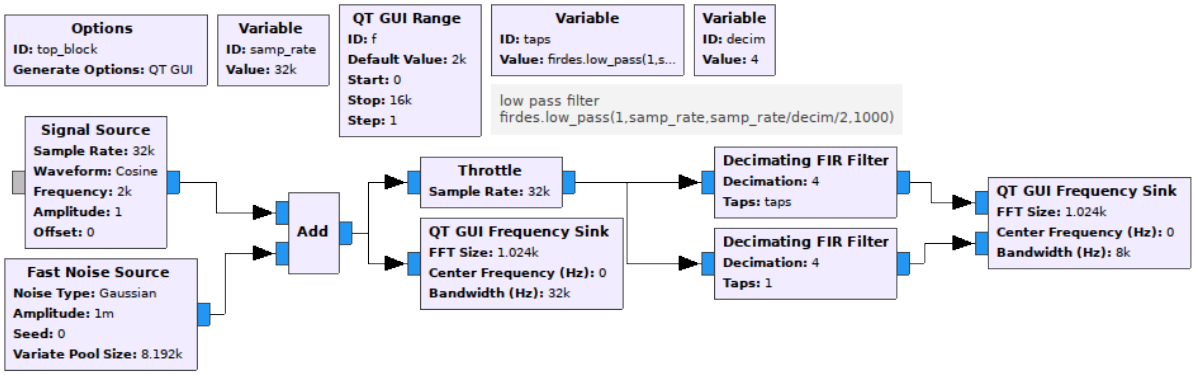

Figure 6: Signal processing chain for illustrating the impact of filtering on the decimation result. Notice that the decimation on the top chart uses a low-pass filter defined through its `taps` whose definition is given as a comment under the associated variable block, while the bottom chart implements the same operation without filtering (`taps=1` meaning that $y_n = x_n$ or an output equal to the input).

purpose Software Defined Radio (SDR) source, we demonstrate here receiving multiple broadcast FM stations simultaneously (Fig. 8) by processing datastreams generated by Analg Devices' PlutoSDR (85 euros from Mouser, reference 584-ADALM-PLUTO).

Despite having already described such an application in [5], we use the opportunity of this new sub-100 euro SDR platform to broaden our horizons to a wider bandwidth. Indeed, the Pluto provides a 10 MHz bandwidth (even 50 MHz when configuring the receiver to the higher grade model AD9364, even if only the lower grade AD9363 is fitted on the board [2]) and is perfectly integrated as a GNU Radio peripheral with a continuous datastream transferred over the USB bus for a sampling rate up to 3.5 Msamples/s. In order to use this peripheral, compile `libad9361-iio` (`github.com/analogdevicesinc/libad9361-iio.git`) and then `gr-iio` (`github.com/analogdevicesinc/gr-iio.git`) and finally use in GNU Radio Companion the newly created source `PlutoSDR Source`. Remember to add the user to the `plugdev` group in `/etc/group` if the provided `udev` rule is to be used.

A 3.5 MHz bandwidth is enough to receive a bandwidth covering 8 broadcast FM stations since the spacing between two stations is 400 kHz. We wish to listen simultaneously to all these stations, not to report to the SACEM agency which music is being broadcast by which station [6] but simply to demonstrate the ability to process all the data collected in the investigated band. Listening to a

[2] The procedure is officially described on the Analog Devices web site at `wiki.analog.com/university/tools/pluto/users/customizing`

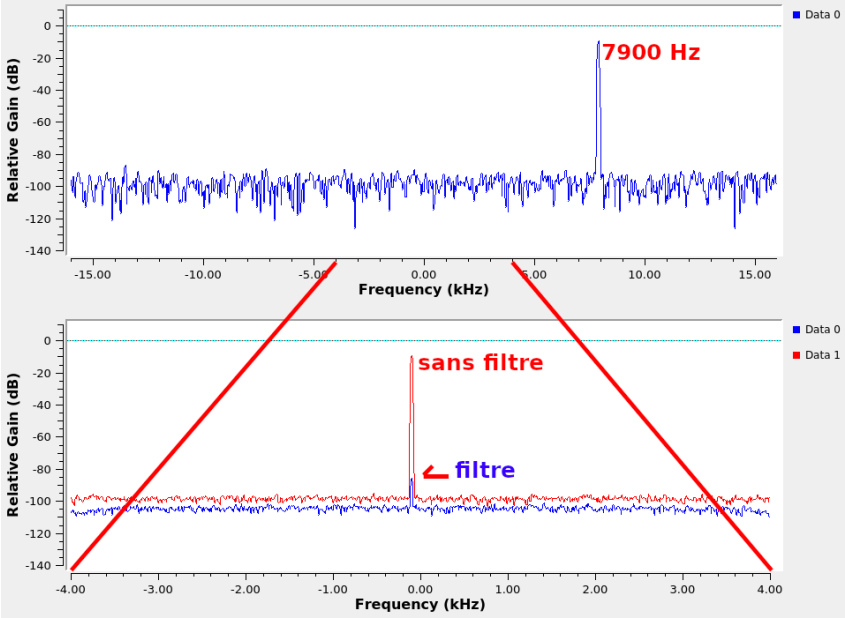

Figure 7: Result of decimating a complex 7900 Hz signal, sampled at 32 kHz and decimated by 4, aliased close to baseband if no filtering is applied (red). The filter (blue) prevents such problems, which practically extend from the discrete spectral component to the continuous receiver noise background. The transposition frequency is null so the `Xlating FIR filter` is not used here, but it allows combining frequency transposition and decimation to get familiar with the impact of filtering when applying these two processing steps.

single broadcast FM station is the first exercise performed by any SDR implementation and is trivially achieved using the processing chain shown in Fig. 8, allowing to check the proper operation of the PlutoSDR source. Having validated this step, we copy a `Xlating FIR filter` frequency transposition block fitted with a "wisely" tuned low-pass filter designed with a broad enough transition width in order to induce few enough coefficients to keep the necessary processing power low while still rejecting the signal from adjacent bands. Hence, we have used a filter defined (`Low Pass Filter Taps`) with a 150 kHz transition band, or about 3500/150=24 coefficients. The processing power needed to compute the convolution with so few coefficients is low enough for a single processor to compute the dataflow for multiple stations: an Intel i5-3320M processor clocked at 2.60 GHz as found on a Panasonic CF-19 computer is enough to simultaneously process all 8 stations available in the analyzed bandwidth. Executing the the processng flow shown in Fig. 9 yields the results shown in Fig. 10, whose sound can be heard at `jmfriedt.free.fr/fmpluto.ogv`. On this video, we have voluntarily initially selected too narrow a filter transition width in order to induce an excessive number of coefficients, requiring more computational power than available to continuously process the datastream from the 8 stations. Broadening the transition width solves the issue by reducing the number of coefficients.

The many sliders are used to tune the audio level of each station and hence select which one or the other is heard on the output of the sound card.

5.2 Exploiting the 2.4 GHz WiFi band for RADAR applications

A second case of applying the frequency transposition requiring the Hilbert transform is the acquisition of signals for a RADAR measurement. WiFi is part of the signals constantly surrounding us: measuring the direct (reference) signal and the reflected (surveillance) signal, possibly frequency shifted by the Doppler effect if the target is moving, allows for detecting targets in the environment of the emitter [7]. The WiFi signal only covers about 15 MHz (IEEE 802.11g) around the carrier, which is for example 2.422 GHz for channel 3. While a PlutoSDR provides the complex signal generated by mixing with a local oscillator and its copy shifted in quadrature (I/Q detector), such a circuit becomes more complex as the frequency

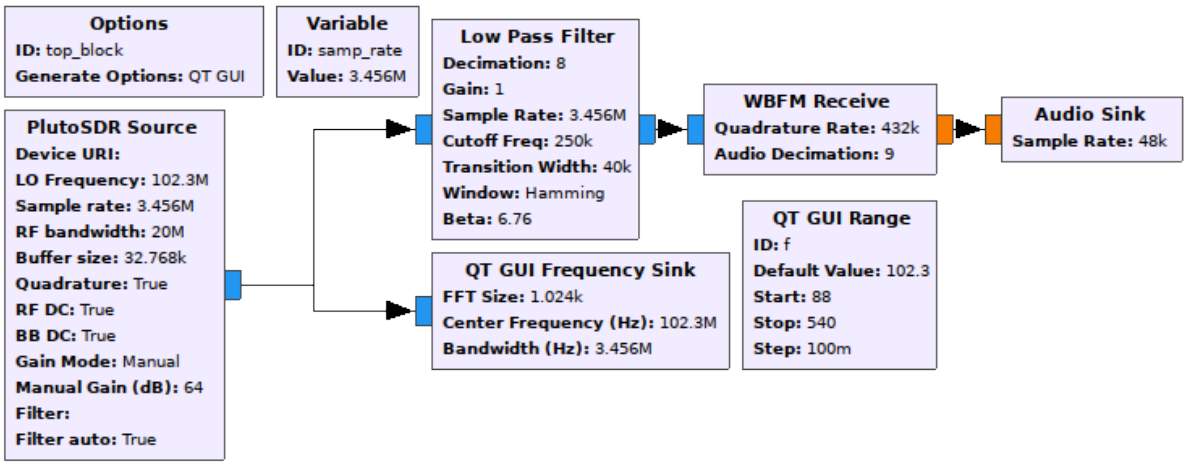

Figure 8: Signal processing chain to listen to a single broadcast FM station using Analog Devices' Pluto receiver. Notice that the synthetic signal source was replaced with a physical data source, hence requiring the removal of the `throttle` scheduling block.

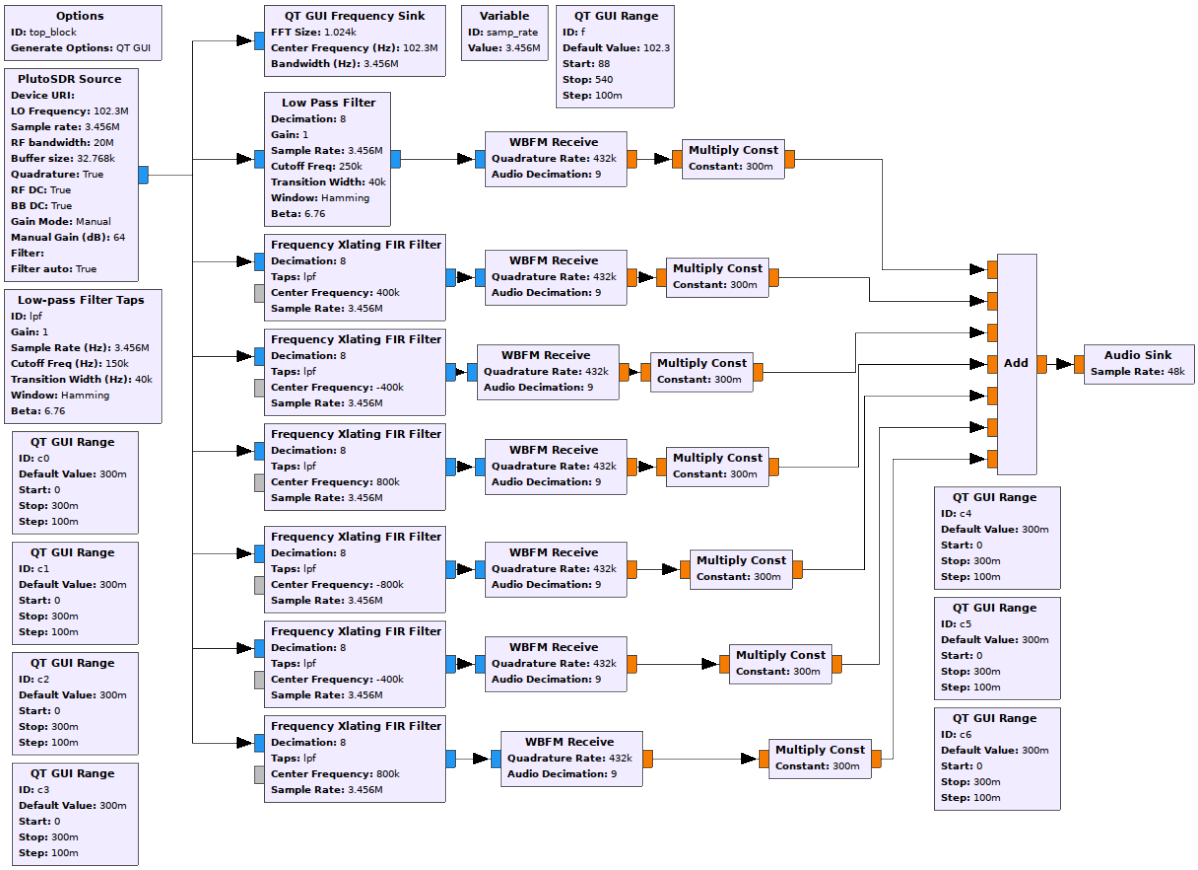

Figure 9: Processing chain to listen to multiple broadcast FM stations simultaneously.

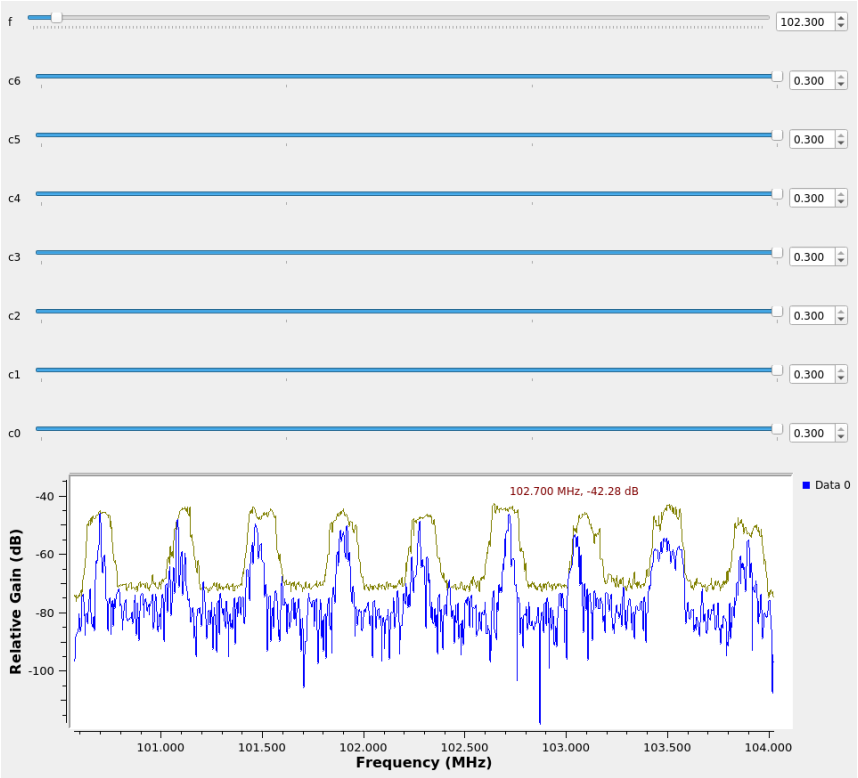

Figure 10: Result of executing the processing chain displayed in Fig. 9. A video dyanmically exhibiting this result can be seen at `jmfriedt.free.fr/fmpluto.ogv`.

is rising and its flaws (I/Q imbalance), namely different gains on the identity and quadrature channels, and a phase different from the ideal 90° between the two channels) more obvious. Furthermore, if we wish to collect signals by using an oscilloscope, most such instruments only provide two inputs and not four as would be needed to acquire two complex signals from the two radiofrequency receiver outputs of the reference and surveillance channels needed for a passive RADAR application [8].

An alternative to the I/Q receiver is to use a simple mixer followed by a low-pass filter (practically, the finite bandwidth of the oscilloscope might be enough as a low-pass filter) to acquire a real (as opposed to complex) copy of the radiofrequency signal transposed close to baseband (Fig. 11). However, we have seens that the Fourier transform of a real signal is even: bringing the WiFi signal straight to baseband would mix negative and positive frequency components of the spectrum introduced by the frequency transposition, and prevent after acquisition their separation by filtering. An alternative solution is to work with an intermediate frequency IF lower than half the bandwidth of the oscilloscope. Under such conditions, a first analog frequency transposition is performed by the mixer – eliminating all flaw issues related to the I/Q detector – and then a second digital transposition is performed after the signal was digitized: the mathematical expression $\exp(-j\omega_{IF}t)$ ensures the quadrature between the cosine and the sine and the unity gain. Using a digital implementation of the complex freqency transposition ensures the ideal behaviour of this processing step. However, we have seen that if we transpose by $-\omega_{IF}$ the spectral component located at $+\omega_{IF}$, we bring the signal of interest to baseband, but since a real signal was recorded we also have a second component with a negative angular frequency $-2\omega_{IF}$ which might, if we are not careful, reach the frequency band of interest due to aliasing. Hence, using the Hilbert transform prior to the digital transposition ensures that this spectral component has been eliminated and solves the issue. This approach was used in [9] and [10].

5.3 Airspy receiver

This signal processing technique seems to be used in the Airspy receiver (`airspy.com`) which acquires from a Rafael R820T(2) radiofrequency receiver a signal transfered with an intermediate frequency equal

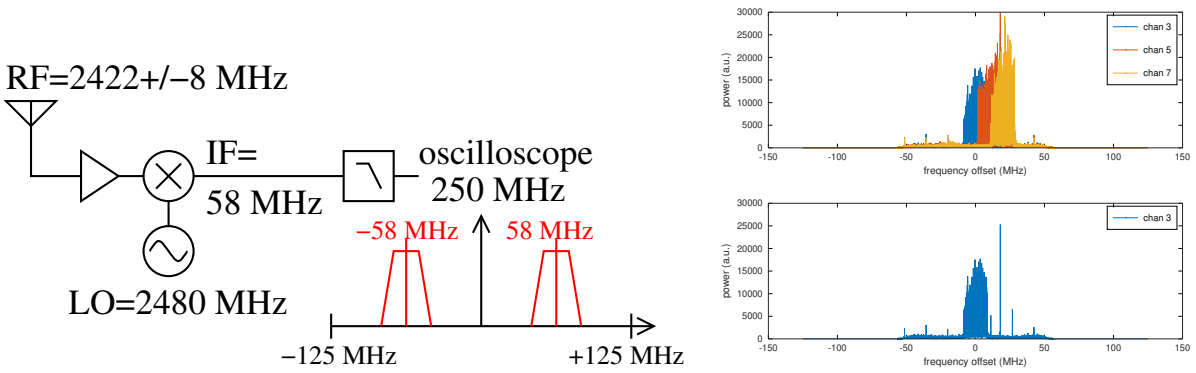

Figure 11: Right: acquisition of signals in three adjacent WiFi frequency bands, combined by summing their spectra computed through their Fourier transform and digitally transposed to baseband (top), and same operation (bottom) on a single WiFi channel (channel 3 centered on 2.422 GHz).

to one quarter of the sampling frequency, before performing a Hilbert transform, filter and decimate on the host computer [3] (Fig. 12, left). Despite analyzing the global processing sequence (Fig. 12, right) by feeding the signal processing chain with a synthetic signal spanning from 0.05 times the sampling rate to 0.4 times the sampling rate (considering that the intermediate frequency is 0.25 the sampling frequency) which demonstrates the proper operation of the whole algorithm, we have not been able to identify each individual processing step by reading the source code, too optimized to allow for isolating each filter, but which nevertheless seems to follow the operations described at [11, pp.210–214]: as observed in `filters.h` of `airspy`, the even coefficients (real values) of the filter are all null except for the central coefficient (called `hbc` in `airspy` to normalize the gain of both branches), and the odd coefficients represent the impulse response of the Hilbert transform [11, Fig 8.10] implemented as a *Half Band Centered* filter. We also find in this reference the demonstration justifying why the filter coefficients are only applied to the samples with odd index of the measurement [11, Fig 8.14].

Since the signal processing chain does not seem to be explicitly documented currently, we have tested by replacing the content of `airspy.c` available at `github.com/airspy/airspyone_host/tree/master/libairspy/src` with:

```
#include <stdio.h>
#include <math.h>
#include "iqconverter_float.h"
#include "filters.h"

#define N (256*64)

void affiche(float *d)
{int c;for (c=0;c<N;c++) printf("%f␣\n",d[c]);}

int main()
{int c;
 float output_buffer[N],freq;
 iqconverter_float_t *cnv_f;

 cnv_f=iqconverter_float_create(HB_KERNEL_FLOAT,HB_KERNEL_FLOAT_LEN);
 iqconverter_float_reset(cnv_f);

 for (c=0;c<N;c++)                        // signaux de synthese ...
    output_buffer[sample_count]=cos(2*M_PI*0.205*(float)c);
 for (freq=0.05;freq<0.4;freq=freq+0.01) // ... somme de sinusoides
    for (c=0;c<N;c++)
       output_buffer[c]+= (freq*3.)*cos(2*M_PI*freq*(float)c);
 affiche(output_buffer);                  // affiche valeurs initiales
```

[3] `opensourceradiotelescopes.org/pipermail/members_opensourceradiotelescopes.org/2018-April/000271.html`

```
25
26  iqconverter_float_process(cnv_f, (float *)output_buffer, N);
27  sample_count /= 2;
28
29  affiche(output_buffer);                  // ... et apres traitement
30  return(0);
31 }
```

in order to feed the processing chain with our own signals with known spectral characteristics. It is furthermore interesting to display the results of intermediate steps of the computation as found in the `translate_fs_4()` function of `iqconverter_float.c` (Fig. 12, right), remembering that the first step of the computation which multiplies the real dataset with the vectors {1, 0, -1, 0} and {0, 1, 0, -1} matches the frequency transposition by the quarter of the sampling rate, converting the real signals to complex values. correspond à la transposition du

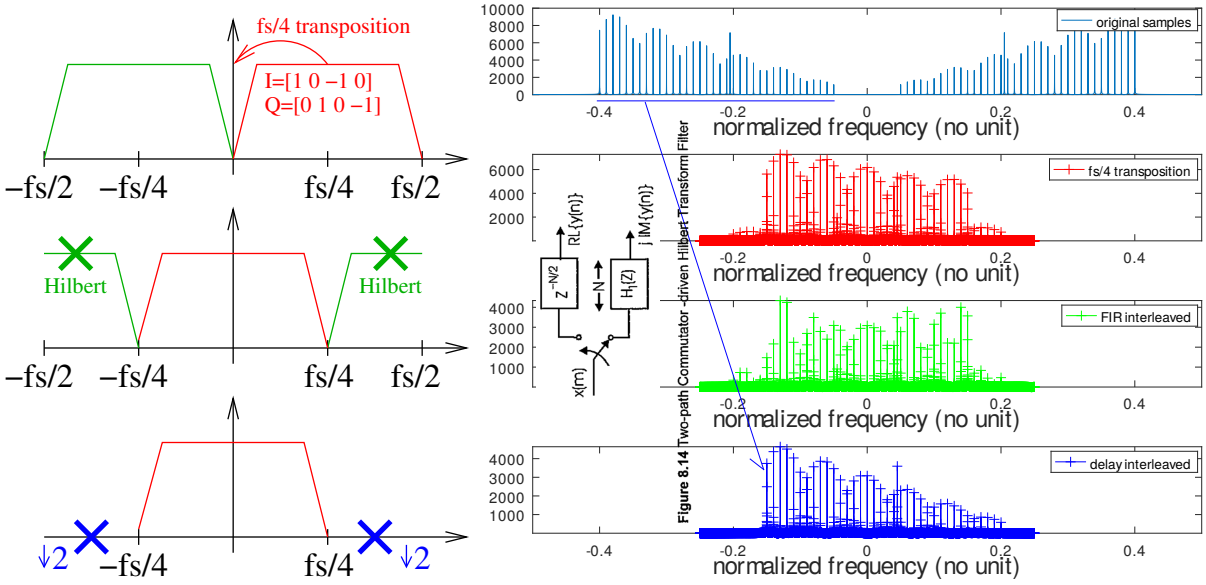

Figure 12: Left: principle of the acquisition of a signal with an intermediate frequency equal to the quarter of the sampling frequency, followed by the frequency transposition to bring the signal to baseband and decimation to eliminate the part of the spectrum no longer necessary. Right: analysis of the implementation of the host side of `airspy` executed on the computer. Notice how the positive part of the spectrum (or negative, since both include the same information since the spectrum of a real signal is even), centered on the quarter of the sampling rate (top) indeed shifts to baseband (bottom) following the processing. The legend of each chart exhibits the name of the function in the source code of the host processing software of `airspy`. Inset: figure from [11] which seems to describe the processing sequence.

6 Conclusion

These exercices using GNU Radio Companion allow for experimenting with some otherwise arid concepts of discrete time digital signal processing when limited to the underlying formal mathematical expressions. We have seen why radiofrequency signals are intrinsically complex values, how to create the imaginary part if the sampling only provides the real part, and how this imaginary part eases further processing, whether frequency transposition or decimation, by removing the negative frequency part of the spectrum early in the processing chain. The Hilbert transform, name of this operation, is however computationally intensive, since it requires a Fourier transform followed by an inverse Fourier transform on the collected datastream.

Following the trend of software defined radio processing and in order to promote its PlutoSDR platform, Analog Devices provides on its web site an excellent book mixing theory, practical demonstrations

on synthetic signals using Matlab (and hence GNU/Octave) and on experiemental datasets [12] which deserves further reading in order to deepen the knowledge introduced in this article.

Acknowledgements

The content of this presentation was assembled as part of tutorial given during the laboratory sessions of the First French GNU Radio days (`gnuradio-fr-18.sciencesconf.org`). Y. Touil has driven the investigation on the signal processing chain used in Airspy on the host computer. All references not freely available on the internet have been fetched at the Library Genesis at `gen.lib.rus.ec`, an invaluable resource for our research and teaching work.

References

[1] J.-M Friedt *& al.*, *Software defined radio decoding of DCF77: time and frequency dissemination with a sound card*, Radio Science **53** (1), 48–61 (Jan. 2018)

[2] M.T. Taner, F. Koehler & R.E. Sheriff, *Complex seismic trace analysis*, Geophysics **44** (6), 1041–1063 (1979)

[3] *Hilbert Transform Design Example*, `www.dsprelated.com/freebooks/sasp/Hilbert_Transform_Design_Example.html`

[4] L.R. Rabiner & B. Gold, *Theory and application of digital signal processing*, Prentice-Hall (1975)

[5] J.-M Friedt, *La réception de signaux venus de l'espace par récepteur de télévision numérique terrestre*, OpenSilicium **13** (Dec 2014/Jan-Fev 2015), available at `jmfriedt.free.fr/sdr2.pdf`

[6] B. Happi Tietche, *Proposition d'architectures radio logicielles FPGA pour démoduler simultanément et intégralement les bandes radios commerciales, en vue d'une indexation audio*, PhD Université Pierre et Marie Curie – Paris VI (2014) as part of the SurfOnHertz project

[7] H. Guo *& al.*, *Passive radar detection using wireless networks*, International IET Conference on Radar Systems (2007), ou K. Chetty *& al.*, *Through-the-Wall Sensing of Personnel Using Passive Bistatic WiFi Radar at Standoff Distances*, IEEE Trans. Geoscience & Remote Sensing (2012)

[8] J.-M Friedt, *RADAR passif par intercorrélation de signaux acquis par deux récepteurs de télévision numérique terrestre*, GNU/Linux Magazine France **212** pp.36- (Fév. 2018)

[9] J.-M. Friedt, G. Goavec-Merou, G. Martin, W. Feng & M. Sato, *Passive RADAR acoustic delay line sensor measurement: demonstration using a WiFi (2.4 GHz) emitter and WAIC-band (4.3 GHz)*, Proc. WiSEE (2018), available at `jmfriedt.free.fr/wisee2018.pdf`

[10] W. Feng, J.-M Friedt, G. Goavec-Merou, M. Sato, *Passive radar delay and angle of arrival measurements of multiple acoustic delay lines used as passive sensors*, soumis IEEE Sensors (2018)

[11] F.J. Harris, *Multirate signal processing for communication systems*, Prentice-Hall (2004), ou N. Robertson, *Simplest Calculation of Half-band Filter Coefficients* (2017) at `www.dsprelated.com/showarticle/1113.php`

[12] T.F. Collins, R. Getz, D. Pu & A.M. Wyglinski, *Software-Defined Radio for Engineers*, Artech House (2018), available at `www.analog.com/media/en/training-seminars/design-handbooks/Software-Defined-Radio-for-Engineers-2018/SDR4Engineers.pdf`