POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

SDR-based automatic RF component analyzer

Tesi di Laurea Magistrale in Telecommunication Engineering - Ingegneria delle telecomunicazioni

Author: **Alessandro Andrea Vogrig**

Student ID: 10527163
Advisor: Prof. Matteo Oldoni
Academic Year: 2024-25

Abstract

In an always connected world, access to information has become trivial. For certain fields, such as telecommunication engineering, the practical aspect of learn-by-doing cannot be neglected, even though it is often limited by hardware inaccessibility. This thesis presents the project of an affordable and extensible platform for radio frequency (RF) components analysis, designed with widely available and affordable software-defined radios (SDR) hardware, to demonstrate that meaningful RF measurements can be performed in low budget scenarios, effectively addressing the scarcity of affordable RF test equipment.
The designed system comprises a HackRF SDR as test signal generator, a Nooelec Nesdr SDR as acquisition interface, both handled by a purposely written software. This work presents an in-depth hardware characterization of both SDR units, including tests for frequency response, gain linearity, intermodulation distortion (IIP3), input impedance (S11), frequency stability, and power handling. Special attention is given to identifying nonlinearities and their sources with the aim of calibrating these imperfections through specifically developed software and using external precision instruments[22][30][31].
The software used in the hardware characterization required the control of complex instruments such as an arbitrary waveform generator and a continuous wave (CW) generator remotely. This led to the development of control scripts through the VISA interface[17], that were used as a base for a quick-start guide for future users of the instruments.
On the software side, a modular, object-oriented framework was developed in MATLAB. The software allows users to generate signals, acquire data, perform power corrected spectral analysis, and run analysis routines such as two-tone intermodulation tests, digital modulation analysis (e.g., 16-QAM) and power response measurements. The software was developed keeping as key objectives modularity and expandability, thanks to the use of software design patterns[6], to allow users to easily add new functionalities to the system. A major contribution of this thesis is the development of a new HackRF interface for MATLAB, that overcomes existing limitations in community-developed libraries, enabling also other developers to integrate the HackRF in their MATLAB code.
The project is released as open-source, and all design files and code are made publicly available to encourage further adoption and development. The work concludes with sug-

gestions for future improvements and a discussion of the results obtained.
The outcome of this research activity hence provides a flexible platform for characterization of radio frequency components, with potential extension to more high-performance hardware units (used as transmitting and acquisition elements) and therefore supports the workplan of the DREAMS project, funded by the European Union under the Italian national Recovery and Resilience Plan (PNRR) of NextGeneration EU, partnership on "Telecommunications of the Future" (PE00000001 - program "Restart", Structural Project DREAMS.

Keywords: software-defined radio, RTL-SDR, RF components tester, spectrum analyzer, low-cost instrumentation.

Abstract in lingua italiana

In un mondo sempre connesso, l'accesso all'informazione è ormai scontato; tuttavia, in alcuni ambiti come l'ingegneria delle telecomunicazioni, l'aspetto pratico del "learning-by-doing" non può essere trascurato, sebbene sia spesso limitato dall'inaccessibilità dell'hardware.
Questa tesi presenta il progetto di una piattaforma economica ed estensibile per l'analisi di componenti in radiofrequenza (RF), realizzata utilizzando hardware SDR (Software Defined Radio) ampiamente disponibile ed economico, con l'obiettivo di dimostrare che misure RF rilevanti possono essere eseguite anche in contesti a basso budget, affrontando la scarsità di strumenti di test accessibili.
Il sistema progettato è composto da un HackRF SDR come generatore di segnali di test e da un Nooelec Nesdr SDR come interfaccia di acquisizione, entrambi gestiti da un software appositamente sviluppato.
Il lavoro presenta una caratterizzazione approfondita dell'hardware impiegato, includendo test di risposta in frequenza, linearità di guadagno, distorsione da intermodulazione (IIP3), impedenza di ingresso (S11), stabilità in frequenza e gestione della potenza. Particolare attenzione è dedicata all'identificazione delle non linearità e delle loro cause, utilizzando strumenti di precisione[22][30][31], con lo scopo di calibrare tali imperfezioni tramite software dedicato.
Il software necessario per la caratterizzazione ha richiesto il controllo remoto di strumenti complessi come generatori di forma d'onda arbitraria e generatori a onda continua (CW), portando allo sviluppo di script operanti attraverso l'interfaccia VISA[17], a partire dai quali è stata redatta una guida rapida per i futuri utenti.
Dal lato software, è stato realizzato un framework modulare ad oggetti in MATLAB. Il software consente la generazione di segnali, l'acquisizione dei dati, l'analisi spettrale calibrata in potenza e l'esecuzione di test come la misura dell'intermodulazione con test a due toni, l'analisi di modulazioni digitali (es. 16-QAM) e la misura automatizzata della curva di risposta in potenza di un amplificatore. L'architettura è stata progettata con particolare attenzione alla modularità e all'espandibilità, mediante l'uso di software design pattern[6], così da permettere ad utenti che ne trovassero bisogno di integrare facilmente nuove funzionalità. Un contributo rilevante della tesi è lo sviluppo di una nuova inter-

faccia HackRF per MATLAB, che supera le limitazioni delle librerie esistenti sviluppate dalla community, consentendo anche ad altri sviluppatori di integrare HackRF nei propri progetti MATLAB.
Il progetto è distribuito in formato open-source e tutti i file e il codice sono pubblicamente disponibili per favorirne l'adozione e l'ulteriore sviluppo. Il lavoro si conclude con una discussione sui risultati ottenuti e con alcune proposte per miglioramenti futuri.
Il risultato di questa attività di ricerca fornisce quindi una piattaforma flessibile per la caratterizzazione di componenti a radiofrequenza, con potenziali estensioni a unità hardware ad alte prestazioni (utilizzate come dispositivi di trasmissione e acquisizione) e supporta pertanto il piano di lavoro del progetto DREAMS, finanziato dall'Unione Europea nell'ambito del Piano Nazionale di Ripresa e Resilienza (PNRR) italiano di NextGeneration EU, partnership "Telecomunicazioni del Futuro" (PE00000001 – programma "RESTART", Progetto Strutturale DREAMS).

Parole chiave: software-defined radio, RTL-SDR, tester componenti RF, analizzatore di spettro, strumentazione a basso costo.

Contents

Rationale and motivations

0.1. Scope

Approaching a new subject like the ones typical of the telecommunications world, as an example, RF systems design or digital communications, usually remains a highly theoretical matter for a student: that is because of the great barrier posed by the availability of affordable instruments needed to access the practical world of experimentation.
Even when those instruments are available, say, as an example, in the case of a university laboratory, their number is not enough to let everyone have the proper opportunity to test what was learned for a lack of time. Moreover, the high cost and fragility of these apparatuses means that a student has to be particularly careful in his experimenting, maybe avoiding verifying some theory for the fear of breaking something.
For these reasons, it is rather apparent why an easily affordable setup would be highly useful, especially in the academic and hobbyist world.
The research work presented in this thesis aims to fill the gap among commercial instruments, giving a possibility to everyone to access a system to analyze and generate signals and perform measurements.
Comparing what market offers, one of the most affordable entry level spectrum analyzer, the Siglent SSA3021X[24] still costs around 1200€, that is not that big of a figure when talking about spectrum analyzers with digital decoding capabilities. On the other hand, looking at the DIY solutions, the Adalm Pluto by Analog Devices can be adapted to be an affordable entry level device. Unfortunately, this hardware still costs about 300€. Can we do better? As one might already guessed, the answer is "yes": by employing open hardware such as the HackRF SDR and an RTL SDR, we can cut the Adalm Pluto cost in half. Moreover, the latter solution has the benefit of not being a monolithic chip as the Pluto is, meaning that breaking the receiver side of the system, by, as an example, providing too much input power, as can happen with an inexperienced student, does not require to replace the whole hardware, but just the receiver device.
Nevertheless, the companion software, that was written for this thesis, was designed keeping in mind the possibility of using different types of hardware, facilitating their in-

tegration.
Other devices are available on the market, like the tinySA, but they are not as cheap or versatile as the proposed solution.

0.2. Structure

To correctly realize the proposed system, a deep understanding of the used hardware is necessary. Key parameters such as linearity, spectrum purity, frequency ranges of the device, maximum usable bandwidth and maximum output and input power must be characterized and methods to correct eventual non idealities found must be developed.
The work was divided in different phases: the reader will find in the next chapters a description of every step, detailing the considerations and the theory behind each design choice, ordered in a logical fashion.
In the first chapter, the general key aspects will be discussed. In the second chapter, the focus will be posed on the receiver device that is used to sample the RF signal to be analysed. The third chapter will describe the transmitter device, that will be used as signal generator to create test patterns for various measurements. The fourth chapter will describe the software created to perform the objectives of this thesis work. The thesis work will conclude with chapter 5, dedicated to discussing the overall results obtained and further improvements.
All the written code is available on a public repository on Github at the URL: `https://github.com/Vogs27/CotSA`.

1 | Introduction

1.1. The importance of spectral analysis in practical RF systems

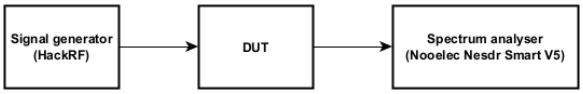
In communication systems, distortion refers to any unintended alteration of a signal during its transmission or amplification. All electronic amplifiers, by their very nature, introduce some level of distortion due to non-ideal behaviours. While this effect is especially noticeable and undesirable in audio applications, where even small imperfections can negatively impact listening quality, its consequences in radio frequency (RF) systems are broader and more complex.

Unlike in audio systems, where the goal is typically to ensure clean and natural sound reproduction, RF systems must also account for efficient use of the electromagnetic spectrum. In addition to maintaining signal quality, RF designs must prevent interference with nearby frequency bands and comply with strict regulatory limits on out-of-band emissions.

A major source of interference in RF systems comes from non-linearities in components such as amplifiers. These non-linear effects can cause unwanted frequencies to appear outside the intended transmission band, creating what are known as spectral regrowth or spurious emissions. In practical terms, this can mean disrupting nearby communication channels or violating transmission standards, especially critical in systems transmitting at high power levels, where even small distortions can translate into significant radiated energy.

To mitigate these issues, RF systems often use filtering techniques to suppress harmonics and other unwanted signal components. For transmission schemes that maintain a constant signal envelope, such as frequency modulation (FM), these filters are generally effective in keeping emissions within acceptable bounds. However, for modulation schemes that involve changes in amplitude, like amplitude modulation (AM), single-sideband (SSB), or many digital formats, such as quadrature amplitude modulation (QAM), non-linearities also generate intermodulation distortion (IMD) close to the intended frequency. These distortion products are difficult to remove through filtering alone, as they lie too close to

the desired signal spectrum[8].
In such cases, maintaining amplifier linearity becomes essential. Ensuring linear performance helps preserve both spectral cleanliness and signal integrity. For this reason, the design of RF systems must carefully balance power efficiency, complexity, and the need for linear operation to comply with technical and regulatory demands.
Besides a careful design on paper, it is apparent the necessity of finding a way to practically assess the performance of the design and its components. For this reason, the use of a spectrum analyser is fundamental: a RF system or part of it, can be tested with external stimulus recreating its working conditions and its response can be measured, effectively checking the adherence of the design to the project requirements.
This is usually done with specialized instruments, called spectrum analysers, where a signal generator produces a test signal that is then fed to the device under testing (DUT). The output of the DUT is then sampled by a spectrum analyser and the spectral content is then displayed to the user to assess the performances. In figure 1.1a its possible to observe a typical connection diagram of a DUT, while in figure 1.1b a testing setup composed with the hardware of this thesis is presented.



(a) Device connection scheme

(b) A device tested with CotSA

Figure 1.1: DUT testing setup.

1.2. A comparison with already existing instruments

As discussed at the beginning of this text, the aim of this work is to build an instrument as affordable as possible but with the characteristics as close as possible of commercial ones. This still is a pretty broad description: what we are actually looking for? As reference instruments, two devices were considered: the tinySA and the Siglent SSA3021X. Both of them were intended as spectrum analysers with some enhanced features: the tinySA can provide a mean of generating an arbitrary test signal, but this signal cannot be used together with the spectrum analyser functionality as they share the same RF path. Considering this in addition to the fact that the tinySA uses a standard ISM (Industrial, Scientific and Medical) transceiver to perform its measurements, we can say that the tinySA is basically an advanced half duplex transceiver with a wide input bandwidth and some analysis functionalities. The Siglent spectrum analyser has more advanced features, as expected from an entry level professional grade instrument. In particular, it is present a synchronised tracking generator and more advanced analysis mode, such as digital modulation analysis, but the tracking generator is restricted to a sinusoid (one might say it is a chirp, since a tracking generator sweeps the spectrum).
As already pointed out, the tinySA can be defined with some approximation as an enhanced half duplex transceiver, as illustrated in figure 1.2 by its own block diagram[7].

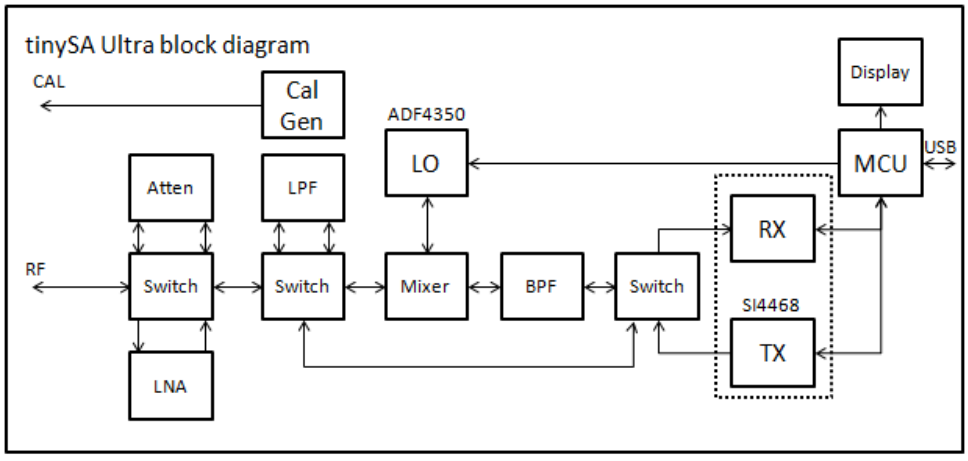


Figure 1.2: TinySA block diagram.

As can be observed, the heart of the system is the Silicon Labs Si4468 transceiver IC in an heterodyne configuration, using a mixer driven by an Analog Devices ADF4350 synthesiser to up/downconvert the RF signal to an IF suitable for the transceiver chip: as we will see shortly, a similar approach will be adopted in this thesis work implementation. As one might imagine, the Siglent SSA3021X has a far more specialized analog section

and frontend, reaching better performances. Its block diagram is not publicly available, but some approximative reverse engineering has been carried out[4]. The SSA3021X employs a dual IF (dual conversion) superheterodyne scheme; the tracking generator tap off its frequency reference from the local oscillator of one of the IF stages, so it is synchronised with the receiver stage. In the conceived design, this feature has proven impossible to integrate without greatly adding complexity to the system; despite that, a sweeping functionality is present.
Both instruments heavily leverage on an SDR structure to perform their analysis. This supported the idea behind this thesis that using commercial off-the-shelf (hence the name of the software developed, CotSA) SDR could lead to an useful entry-level instrument.
The aim of this work is to prove the feasibility of adopting less specialized hardware to perform measurements. In particular, the work will focus on implementing and proving features with a growing complexity, starting from a simple spectrogram, then adding test with specific test signals, like, as an example, the two tones test.

1.3. What is an SDR?

With the improvements in computational capacity of modern computers, the opportunity of moving part of the signal processing chain from hardware to software presented. This allows a great flexibility in the design of the systems, reducing the complexity of the RF chain and enabling a degree of reconfigurability, since most of the functions are carried out in software rather than in hardware. Exactly as is for analog transceivers, various architectures of SDR exist, meaning that we can find direct-IF architectures or heterodyne architectures[25]. Moreover there exist monolithic SDR transceivers, that embody most of the RF and digital circuit in one integrated circuit (IC), like the Analog Devices AD9363, used in the Adalm Pluto SDR and some Ettus Research SDRs and less integrated architectures like the one used in the Greatscott Gadget HackRF, that will be employed in this work.
The main difference with non-SDR systems stands in the fact that the bulk of signal processing is carried out in the digital domain[23]. As an example that can be easily extended to the transmitter case, an SDR receiver might have the block diagram as in figure 1.3, where, after some signal conditioning, such as image frequency filtering and band tuning with a mixer, the signal is digitalized and further signal processing, such as additional filtering, downconversion, channel equalization and information decoding are carried out in software.

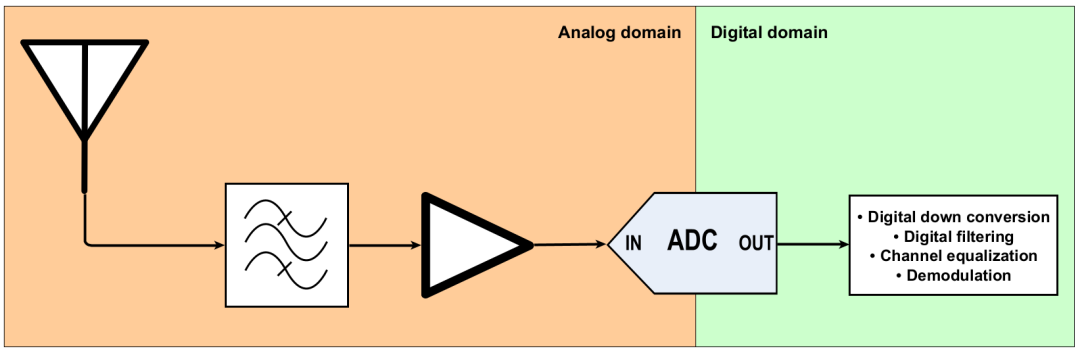
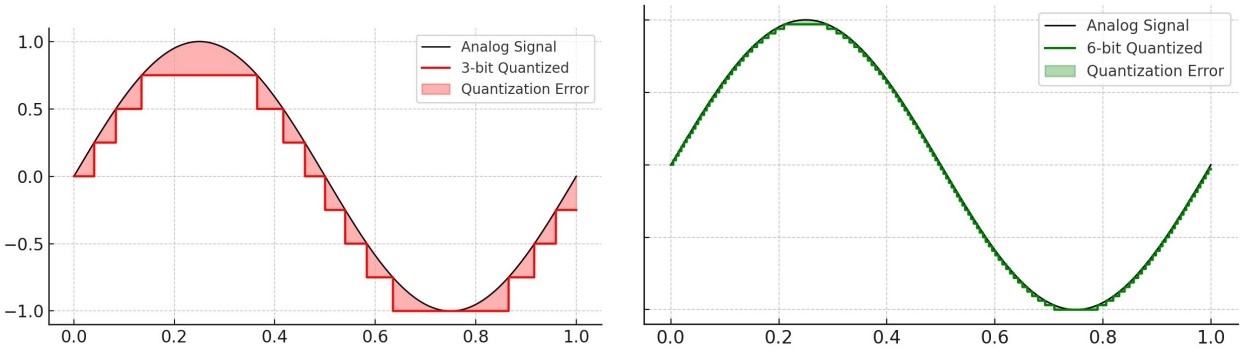


Figure 1.3: Generic SDR receiver block diagram.

Considering this, it becomes apparent how the role of the analog to digital converter (ADC) and digital to analog converter (DAC) is crucial to obtain acceptable results, as these two devices are the interface between the digital world and the real one. Beyond the specific architectures of the two conversion devices, which fall within the domain of electronic engineering, the figures of merit relevant to our application are: the number of bits of the devices, the dynamic range, the spurious-free range, and the allowable sampling frequencies. The sampling frequency directly relates to the maximum bandwidth of the acquired signal, due to the Nyquist limit: $B \leq 2f_s$ when signals are real, and $B \leq f_s$ in case of complex signals, considering B as the bandwidth of the signal and f_s as the sampling rate.

The number of bits of an ADC or DAC refers to the number of bits used for quantizing a given signal, meaning an analog signal can be mapped to 2^{bits} different digital values. This quantization approximate the real signal to a discrete signal adding an error called quantization error. The magnitude of this error, besides the quality of the hardware itself, is also defined by the granularity of the available values, so an higher number of bits corresponds to a reduced quantization error.

The importance of the DAC resolution is relevant to represent the ideal signal as closely as possible: high quantization error introduces spurious into the spectrum of the generated signal. This effect is unavoidable, but it becomes more evident the less the resolution of the used DAC is. On the other hand, the importance of ADC resolution is strictly related to the dynamic range: we can think that having more quantization levels, or discrete values, helps us to better differentiate and space out signals from noise, more closely representing them to actual real world values. In other words the higher the resolution of the ADC, the less is the quantization error, that is the difference between the discrete digital assigned value and the actual value of the measured signal (figure 1.4).



(a) 3 bits quantization.

(b) 5 bits quantization.

Figure 1.4: Comparison between different quantization levels.

The dynamic range represents the ratio between the maximum and the minimum signal that the system can measure. As said before, the dynamic range is given by the ADC resolution, formally:

$$DR_{[db]} = 20log(2^{Nbits}) + 1.76 \tag{1.1}$$

Solving the logarithm, we obtain:

$$DR_{[db]} \approx 6.02 \cdot Nbits + 1.76 \tag{1.2}$$

where the constant term accounts for quantization noise.
A real world system is always affected by noise of various kind. When making a measurement, it can be expected to obtain a reduced range compared to the dynamic range. In this aspect, one can consider the dynamic range as the theoretical best SNR, signal-to-noise ratio, achievable, where the noise only affect the lowest measurable level of an ADC. A way to counteract this undesired effect is to perform oversampling: by capturing a greater quantity of samples than the one required by the Nyquist theorem, the noise can be averaged out among samples. The improvement can be quantified in:

$$SNR_{improvement[db]} \approx 10log(M) \tag{1.3}$$

where M is the oversampling factor.

1.4. Hardware selection

There exist various configuration of SDR systems, but the two most common groups in commercial devices are transceiver SDR systems and receiver only SDR systems. We can further divide the first group in half duplex transceivers and full duplex transceivers: in the first case the RF path is shared among the receiver and transmitter functionalities and the reception and the transmission phases cannot happen at the same time. On the other hand, full duplex transceivers have different RF paths hence the capability to receive and transmit at the same time. It has to be noted that the latter type usually is more expensive than the first one.
While it is tempting to use a single full duplex device to interface with a device under testing (DUT), making the measurement device as compact and integrated as possible, a different approach was chosen. If we reduce the possible devices for this project basing the decision on their cost, that must remain below the cost of a tinySA, one will soon discover that just a few devices are available satisfying this requisite: the Analog Devices Adalm Pluto, a full duplex SDR transceiver; the ANT SDR E200, a full duplex SDR based on the same design of the Pluto; and the GreatScott Gadgets HackRF, an half duplex transceiver SDR, coupled with the Nooelec RTL-SDR, a receiver only SDR. The RF performances of the Adalm Pluto and the ANT E200 are comparable, and the characteristics of the transceiver module are the same, being based on the same chip. On the other hand, the HackRF and the Nooelec SDR are less performant, but their cost combined is around one third of one of the other devices considered. Moreover, most of the cost resides in the transmitter module, the HackRF, resilient enough in this use case, being that even in the worst case scenario of an unterminated port with total power reflection towards the transmitter, the device is resistant enough not to take any damage. This translates into a system where, if an inexperienced users (that is the expected typical user for an entry level instrument like this) inadvertently applies a power level exceeding the maximum allowed at the receiver port, the only damage is taken by an inexpensive part, the Nooelec RTL-SDR, costing around 25€, and easily available on the market. In the other cases, the damage would require a complete replacement of the whole hardware, costing around 250-300€. Unfortunately, selecting two different devices bring another disadvantage: the receiver and transmitter sides work under different clock domains, meaning that they use two different and not synced clock sources. As we will see in 4, this will not be a problem, as the generated and received signals can be aligned in frequency in software, while the coherence of the phase is not a strict requirement, since the measurements do not require a tight relation between transmitted and received phase such as in vectorial signal analysis. For these reasons, the HackRF was chosen as signal generator, while the

Nooelec RTL-SDR was picked as receiver.

2 | Acquisition device

2.1. Device description

As acquisition device, the Nooelec Nesdr smart V5 (figure 2.1) has been selected. The Nesdr smart is based on the Realtek RTL2832U chipset. This chip was initially designed as a DVB-T TV receiver chip and for such reason its architecture presents features typically used in digital television reception, like MPEG video stream hardware packet identifier and so on.

Figure 2.1: A Nooelec Nesdr smart V5.

Unfortunately, the complete history of the discovery of the hidden capabilities of the RTL2832U chip got lost. However, it is known that in 2012, a group of researchers, among them Steve Markgraf, Antti Palosaari, Eric Fry, Dimitri Stolnikov, Hoernchen, Kyle Keen, Christian Vogel and Harald Welte, discovered that the demodulator and decoding stages of the chip can be bypassed (figure 2.5), keeping just the analog-to-digital converter stage directly passing data to the USB interface[36].
This discovery marked the beginning of a revolution in the field of Software-Defined Radio: devices that once costed hundred of dollars, were now available for a few tens of dollars.
The RTL2832U[21] chip represents just half of the reception chain, being a low-IF or baseband (depending on the configuration) analog-to-digital converter. The other fundamental component is represented by the tuner IC, comprising a mixer, the PLL to drive the mixer and the circuitry to filter the generated IF and adjust the reception gain. Over the years, the drivers were expanded to support various tuner ICs from different vendors and with different tuning ranges. Nowadays, the most commonly used tuner is the Rafael

Micro R820T[20], that is also the one present in the Nooelec Nesdr. In some boards the R820T is replaced with a chip marked R860: this change of markings only reflects a marketing decision by Rafael Micro and the actual silicon is the same.
The schematic of the Nooelec device is not publicly available, but datasheets of the two aforementioned chips, that were protected by NDA, eventually leaked onto the internet, so a reference schematic is available. Those schematics will be used from time to time to try to explain some results checking a probable hardware configuration. In figure 2.2 is visible the printed circuit board (PCB) of a Nooelec Nesdr. The coloured boxes identify the various components comparing them with a schematic in figure 2.3.

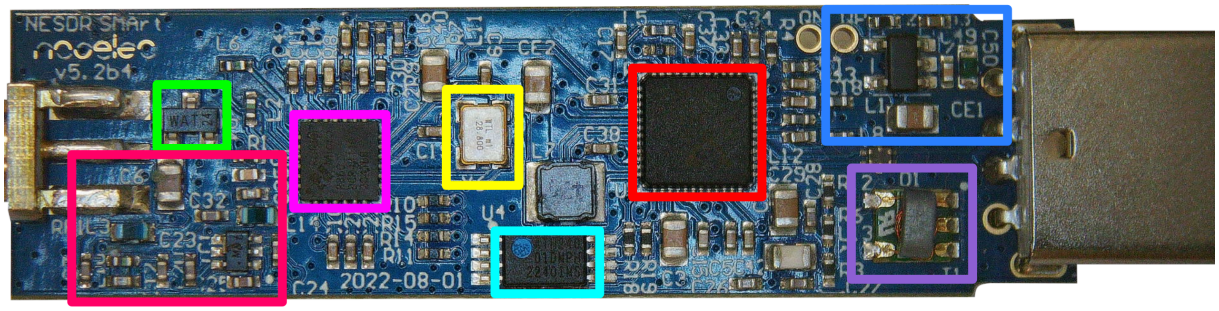


Figure 2.2: Nooelec Nesdr PCB. Light green: ESD protection diode. Magenta: input matching circuit. Pink: R820T tuner. Yellow: crystal oscillator. Light blue: EEPROM. Red: RTL2832U. Blue: power supply. Violet: direct sampling balun.

The schematic in figure 2.3 closely relates to the actual Nooelec design, as it was designed by the author using both the reference designs of the two used chips and a reverse engineering of the actual hardware, although the reference designators of components aren't matched and some minor differences in components value and topology might be present. We can observe marked in yellow the 28.8MHz temperature compensated crystal oscillator that generates the clock for both the RTL2832U IC (marked in red) and the R820 IC (marked in pink): the crystal signal is acquired by the R820T chip and than amplified with a buffer stage and passed directly to the RTL2832U. The EEPROM memory marked in light blue stores the configuration data for the RTL2832U. The power section, comprising some filtering and a voltage regulator is marked by the dark blue square. Finally, we can see the electrostatic discharge (ESD) protection diode on the RF input marked in green and the rest of the RF path, comprising filters and matching circuits, marked in magenta and violet. The main differences respect to the reference schematic can be found exactly in this last magenta and violet sections: in section 2.2 will be clear why those differences exist. As is for the rest of the components, comparing the Nooelec Nesdr with other RTL based SDR, the difference are little to none.

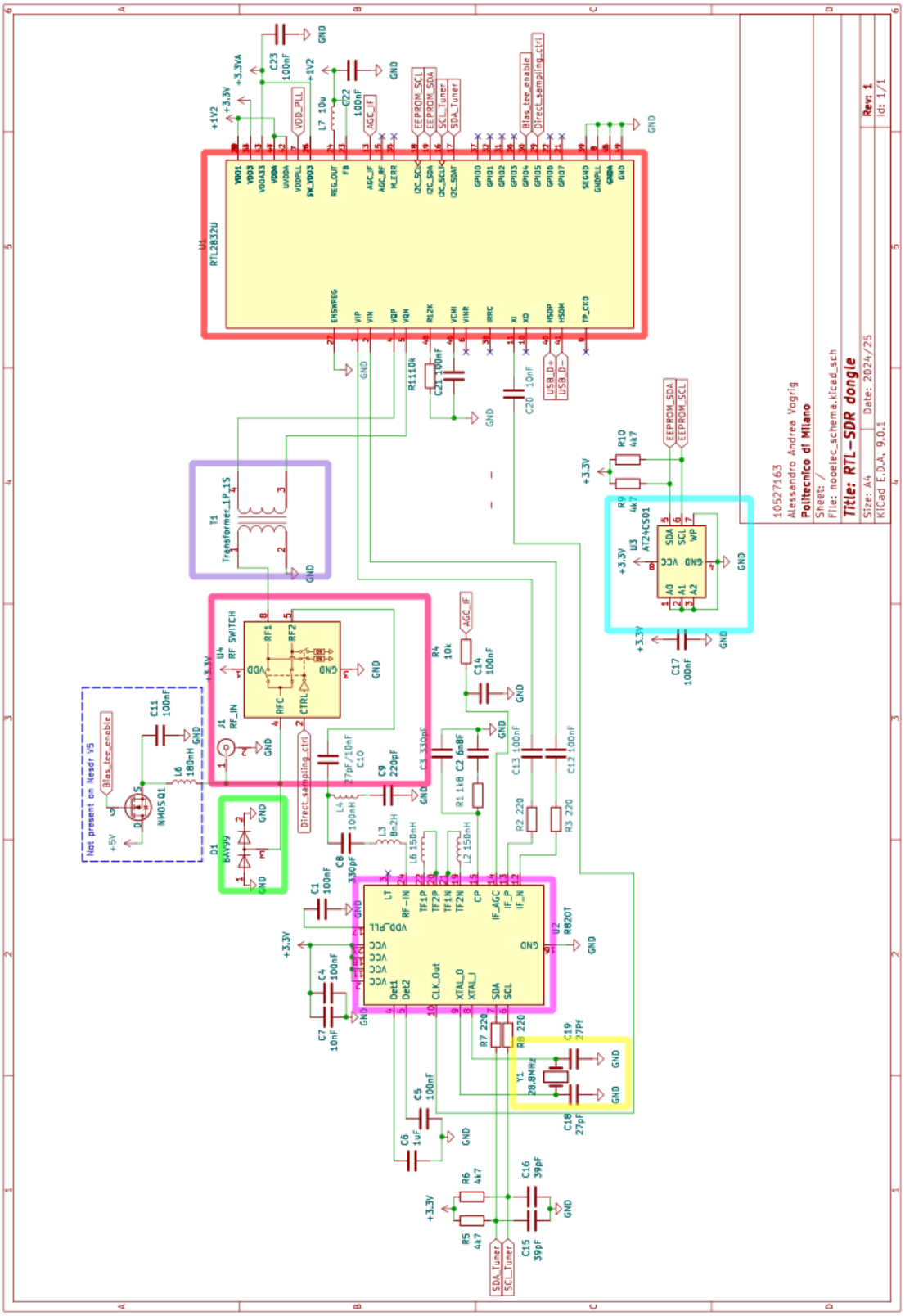


Figure 2.3: RTL-SDR reference schematic, marked with the same color coding of figure 2.2. Power sections omitted.

2.2. Block diagram

In the last section we discussed the schematics of the Nesdr. At the logical level a direct match between the system block diagram and the schematic can be made.

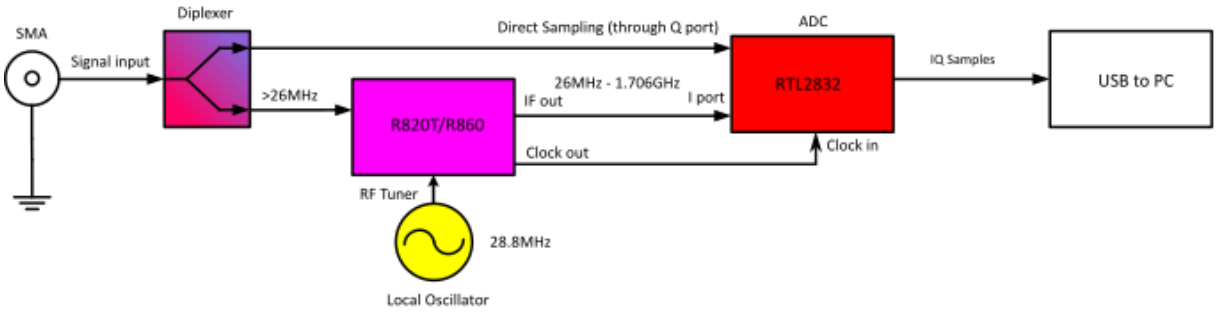
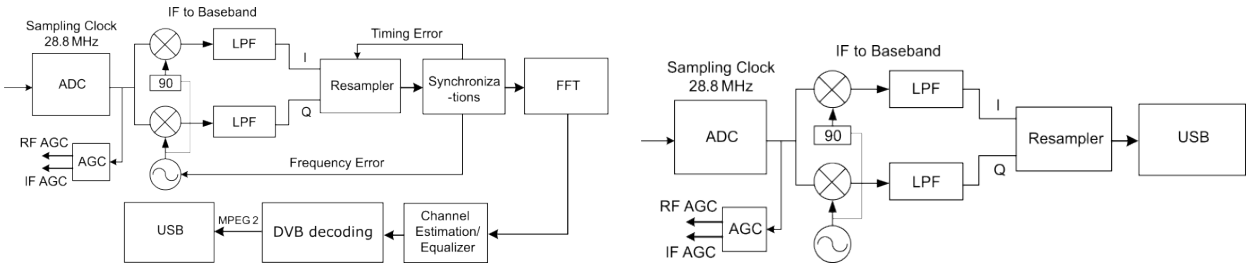


Figure 2.4: Nooelec Nesdr smart V5 block diagram.

The block diagram in figure 2.4 focuses on the signal path, omitting parts not relevant in that regard. Reusing the same colours to highlight the connection between the circuit parts and their functions, we can see how the input signal is divided in two branches: one goes through the tuner IC, while the other, that we won't use in this application, goes directly to the ADC. The RTL2832 is a very versatile chip that can accept a baseband signal divided in I and Q branches, working in a zero-IF mode, or an IF signal. If the latter case is considered, than the two ports on the chip, that would be used for the I and Q signals in the zero-IF case, become two mutually exclusive IF inputs, meaning that the RTL2832 chip can have two different IF inputs to digitalize to choose from. Nooelec has chosen to utilize one of the two port for the IF of the R820T tuner, while routing the other one directly to the RF input. The reasoning behind is that an ham radio operator could use this direct sampling port in conjunction with a proper bandpass filter, to avoid aliasing, to capture the LF (low frequency) band, outside from the supported bands of the tuner.
In theory, this feature could be employed with an external mixer also to receive signals outside the upper limit of the R820T tuner, further extending the capabilities of this project. However this option is disabled in software as is not directly supported by the MATLAB RTL-SDR Toolbox (workaround to solve this issue are available, but the difficulty of applying them can be overwhelming for an inexperienced person).
Another thing that catches the eye is the local oscillator shared between the tuner and the ADC chips: there is not an apparent reason behind this choice, besides keeping the component count thus the costs down and saving space on the printed circuit board.

Looking at the two chips as monolithic blocks does not tell us much: luckily datasheets come to the rescue.



(a) Complete block diagram. (b) Actual stages used.

Figure 2.5: RTL2832U internal block diagram.

The RTL2832U is a complex IC with many built-in functionalities tailored to digital video broadcasting reception and decoding, as visible in figure 2.5a. As visible, the reception chain comprises a 7 bit ADC sampling at 28.8MHz, followed by an AGC (auto gain control) block, that won't be used in this projects, as it would make power readings unreliable. The AGC block scope is to select gains in the tuner's low noise amplifier such that the whole ADC range is used. Unfortunately, the block settings are not directly controllable in software, so there is not a simple way to know how much gain is being used, thus rendering impossible to define a relation between the readings of the ADC and the actual input power at the RF port of the SDR dongle for a lack of information about hardware configuration.

After the ADC block we find the typical scheme of an I/Q demodulator: since the demodulation is being carried out in the digital domain, an I/Q imbalance, that is a problem that arises when the in-phase and quadrature branch have not the same gain and the 90° phase shifter does not produce an exactly 90° phase shift, greatly impairing the quality of the output signal, should not be present.

The now generated I/Q samples cross a resampler block: while the ADC samples at a frequency of 28.8MHz, the output bandwidth of the SDR and its output sample rate are 2.4MHz, this because oversampling is used: more sample than needed (according to the Nyquist theorem) are taken so that they can be averaged in groups to lower the overall noise floor and compensating for the modest ADC resolution.

Going onwards the chain in figure 2.5a, we find a carrier synchronization block, fast Fourier transform block, channel equalizer block and a DVB specific decoding block. Comparing this scheme with the one in figure 2.5b we can notice where the beauty and power of this chip is: most of the processing block can be bypassed, giving the user a raw stream of samples, making possible using this IC as the core of an SDR system.

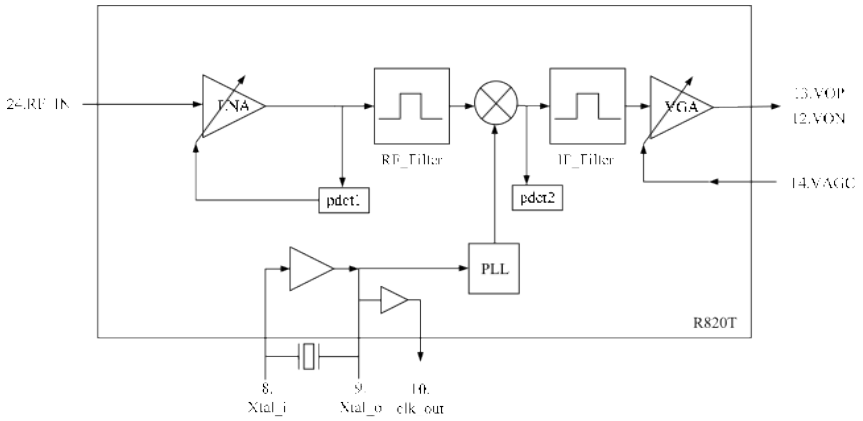


Figure 2.6: R820T block diagram.

The R820T block diagram, based on the datasheet, provides us high level information on the internals of this tuner chip. As we will soon discover in section 2.3, this diagram shows just a simplified part of the whole system. At the beginning of the signal chain an RF-side low noise amplifier can be found. In our application this LNA will remain disabled, as the drivers allows the gain of this amplifier to be controlled only by an internal AGC stage (power detector "pdet1"). The signal, then, passes through a tunable bandpass filter before being downconverted to the IF frequency, which is around 4MHz, by the mixer. The clock used by the mixer is generated by a fractional phase locked loop whose reference clock is given by an external 28.8MHz crystal. After the mixer, an IF filter to remove the image frequency can be found, then another amplifier stage. This amplifier can both be controlled by the RTL AGC output or manually. The latter modality will be the one employed in this project.

2.3. Driver support

The drivers of RTL based SDR are open-source and developed through public contributions to the repository hosted by Osmocom[5]. While it is true that some manufacturers tailor this drivers to enable some particular functions on some specific implementation (think, as an example, dongles with bias-tee power supply at the RF port), the core remains the same. The main library is called rtlsdrlib and employs the WinUSB library to handle data transfers over USB.

Most of the functionalities exposed by the driver come from the reverse engineering of the datasheets. In particular, after reading the R820T registers documentation, is easy to realize how the block provided simplifies the internals of the chip: every single block

has more than one register to deeply configure its parameters: as an example, just the mixer has configurations on the bias current, the clock preamplification and filtering, the output filtering and so on. From this consideration, it is clear how the search for the best parameters can be an endless quest. Moreover, the drivers are compiled to expose just few parameters to the programmer, leaving others hardcoded. For these reasons, it has been assumed that the best trade-off between performance and ease of use in the drivers has already been made by the developers. While some useful changes, like the IF filter bandwidth, could improve the performance of the measurement system studied in this thesis work, the simplicity and ease of using already compiled drivers instead of compiling ad-hoc drivers for the end user was valued more important and can be considered a design choice.

2.4. Optimal working conditions

As discussed in section 2.3, a subset of the whole feature-set is provided with the driver by default, exposing just the most used and most influential parameters. Moreover, the target code for this project is MATLAB, meaning that hardware interaction with the software environment is provided through a dedicated toolbox[35] (that are software add-on in MATLAB terminology). As already discussed previously, it was prioritized the ease of use of the final software out of practicality, instead of deeply tailoring drivers, that would have required the final user to install a custom toolchain to compile them from sources. Following this reasoning, it was chosen to keep the RTL-SDR toolbox from MATLAB as provided. While workarounds invoking directly drivers via C++ external calls and bypassing the MATLAB RTL-SDR Toolbox to access a more reach feature-set is possible, a more sensible approach was chosen, wanting to avoid possible unpredictable side-effects. These choices reduced the available tunable parameters to the following:

- tuned center frequency;
- sample rate, which directly affects the acquired bandwidth;
- IF gain;
- auto gain control enable.

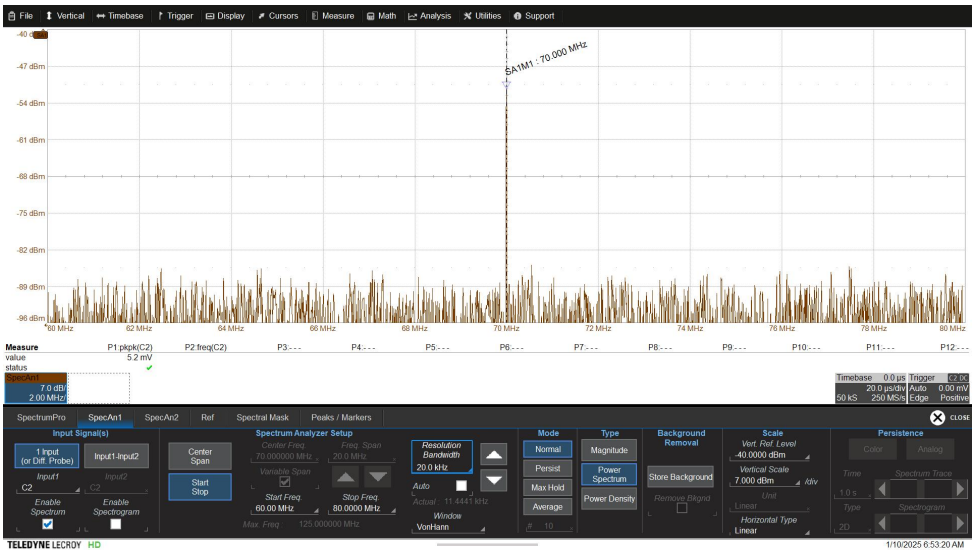
As already discussed, the auto gain control, AGC in short, will not be used, as the input gain (or loss) needs to be note at the software side to correctly calculate the input power and the AGC function does not provide this necessary information.
An experimental characterization of the input gain vs. frequency will be discussed in section 2.5, while a frequency stability analysis will be described in section 2.6.

The RTL chip allows different sample rate to be selected from a predefined list, from 250ksamples/s up to 3.2Msamples/s[1]. These sample rates are dependent on the 28.8MHz clock that is used. As the Nooelec dongle provides I/Q samples, following the Nyquist-Shannon sampling theorem, the acquired bandwidth is equal to the sample rate. As one can notice comparing the results of chapter 3, the bandwidth limit is given by the RTL device, as the device used as signal generator is capable of outputting signals with wider bandwidth than the maximum acquired by the RTL. The question then becomes what is the maximum bandwidth supported by the Nooelec. Since the chip is not used in the way the manufactured intended, a maximum sample rate is not clearly given. Moreover, it is difficult to show, both in an experimental way and even worse on paper, what the differences are among the various available sample rates, because of the nature of these impairments that show off as glitches and misconfiguration of the device giving incorrect readings. In particular, a too high sampling rate leads to a loss of responsivity of the device, meaning, as an example, the device does not react to changes of the tuning frequency. This problem is easy quite easy to detect, as the spectrum displayed on the spectrogram does not change while the tuning frequency is changed. It has to be noted that the device can be used at this high sampling rates, that are above 2.4Msamples/sec, as those glitches occur in a random way. Unfortunately, the only way to exit this lock-up states is an hard reset of the device performed through a power cycling. The absolute sampling rate upper limit is 3.2Msamples/s, above this rate, the device simply becomes too unstable to be used. On the other hand, neither sampling rates lower than 2.4Msamples/s are free of problems: it as been noticed, as an example, that sampling at 2.048Msamples/s leads to a slightly worse noise floor and readings with an higher spurious content. While a clear cause of these problems has not been found nor thoroughly investigated, some logical explanations can be formulated: considering too high sampling rates, a plausible cause can be a loss of packets, thus a loss of configuration parameters, due to latencies in the I/O buffer of both the host computer and the SDR device; as for the inconsistencies found with lower sampling rates, a possible justification resides in clock mismatches that leading to timing errors or lost samples, thus artifacts in the signal acquired.
Both in the practical comparison carried out in this thesis work and in the literature, the best performances are obtained with a sampling rate of 2.4Msamples/s, hence a maximum instantaneous input bandwidth of 2.4MHz. This value has been chosen as the hardware sampling rate. Where a smaller bandwidth is desired, a filtering and downsampling operation is carried out in software, as will be illustrated in chapter 4.

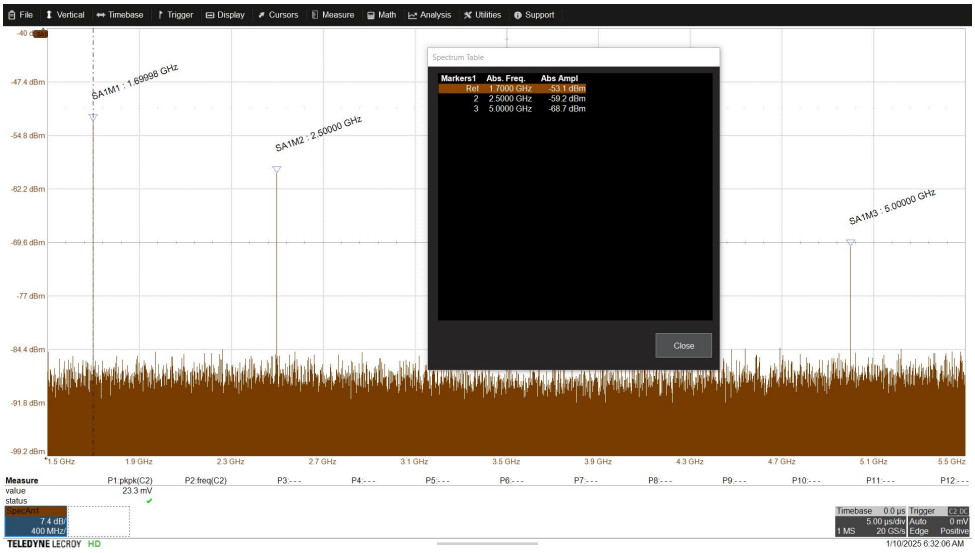
[1]Available sample rate: 250ks/s, 900ks/s, 1.024Ms/s, 1.4Ms/s, 1.8Ms/s, 1.92Ms/s, 2.048Ms/s, 2.4Ms/s, 2.56Ms/s, 2.8Ms/s, 3.2Ms/s.

2.5. Frequency response analysis

As any real world device, some frequency dependent losses are expected in the analog RF/IF path. To obtain not only qualitative measurements, but also to quantify the obtained data, those losses needs to be measured and understood.
The R820T datasheet states a frequency range from 42MHz to 1002MHz, but it does not have any hardware lock to prevent someone configuring the PLL to tune on higher or lower frequencies. The hacking community quickly found out that this range can be extended to cover a range from 26MHz to 1766MHz without particular difficulties. Even wider range extension were tried and some particularly statistically lucky chips capable of working from 13MHz to 1860MHz, tweaking configuration registers and pushing the hardware to its physical limit, where found. Considering this more like an edge-case than the norm, only the range between 26MHz and 1766MHz was taken into account.
Looking at the schematic presented in figure 2.3, besides attenuation due to parasitic capacities of the tuner and non-idealities in the PCB, an effect caused by the input conditioning path is expected. In particular, we can expect a low-pass and high-pass effects by the two input inductors and capacitors, that for certain frequencies can act as resonators thus filters.
The test to assess the frequency response was designed to be also useful as a calibration measurement, as will be discussed in chapter 4. The test setup comprehended the Nooelec Nesdr dongle attached to a PC running the test software and a Rohde and Schwarz SMA100B CW signal generator to generate the test tone (figure 2.8). This calibrated instrument is capable of outputting an high purity sinusoid at a defined power as shown for two test frequencies in figure 2.7, where a spectrum analysis view taken with a Lecroy Wavemaster 8330HD MSO (mixed signal osilloscope) of the test signal is visible.



(a) Tone at 70MHz. Measured power: -50.6dBm.



(b) Tone at 1700MHz. Measured power: -53dBm.

Figure 2.7: SMA100B tones validation. Used power: -50dBm. Cable losses not accounted. Signal acquired with Lecroy WaveMaster 8330HD MSO.

The test consist in a sweep int the frequency range of the RTL dongle at defined power and with a specific fixed configuration of the Nooelec device. For this measurement, instead of a continuos signal with constant changing frequency, like a chirp, a series of sequential sinusoid equally spaced were used to sweep the entire Nesdr frequency range. This was done with various motivations: it relaxes the requirement in term of syncing of operations between the signal generator and the receiver dongle, but, more importantly, it enable to establish a clear relation between the ideal generated tone and the received

one. Using a fine enough tone spacing and applying some interpolation, the results are acceptably precise for the whole range. A study of the SMA100B controls was necessary as the test was automated in its entirety due to the time required to complete it at the desired resolution of 1MHz spacing of the tones. For this reason, the VISA interface[17] of the instrument was utilised, and a program to control the process was written.

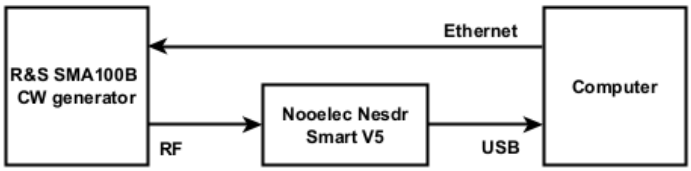


Figure 2.8: R&S SMA100B and Nooelec Nesdr connection setup.

VISA, that stands for Virtual Instrument Software Architecture, is a program interface to control remotely a measurement instrument. It is a industry standard controlled by the IVI Foundation. Despite being a standard, each instrument manufacturer has his "VISA flavour" meaning that while the syntax and the basic commands are common basically for every instrument, each manufacturer choose how to implement a certain function in different ways by series of different and custom tailored VISA commands. To add complexity, VISA is just an application interface (API), that is a collection of commands to enable a program to talk with another, in our case, a test program that talks with the software of the instrument; but it does not specify how this interface should be implemented physically (e.g. connecting to the instrument via USB, LAN, GPIB bus, serial). In our use-case, an Ethernet connection was chosen, so the communication was carried over TCP/IP, using a protocol called VXI-11. Unfortunately, it is not just a plug and play operation, but some drivers are needed. An entire chapter could be dedicated to this topic: suffice it to say that it is a world riddled with poor documentation and broken software, which make it difficult to approach an otherwise highly practical subject. A lifeline is provided by National Instruments, one of the only manufacturers providing reliable and working drivers[14] implementing VISA over various interfaces, also compatible with different manufacturers, so much that many of them directly given up trying develop their own and started recommending National Instrument drivers by default.
The code was developed with MATLAB. A pseudocode representation is given in algorithm 2.1. The idea behind this code is to measure a raw value as read by the ADC for some points (frequencies) in the tuning range of the Nooelec SDR. Since the reference voltage used by the ADC is unknown, also the magnitude of the detected signal is in absolute

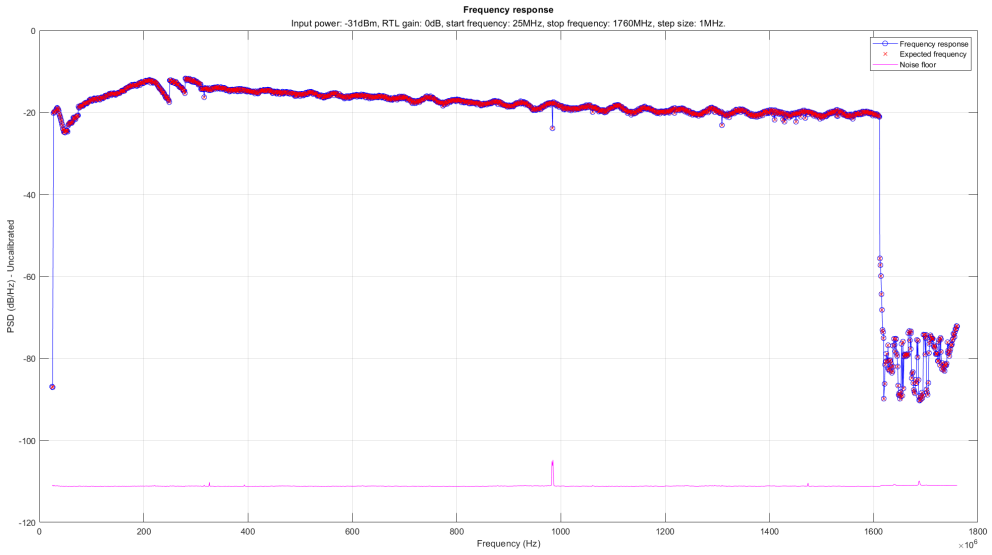
units referred to the ADC full scale, given that a voltage-resolution relation of the ADC cannot be established a priori. This will not be a problem, as the test will be performed at constant power, so effectively any variation in the read magnitude can be interpreted as a variation in frequency response of the device without loss of precision. Moreover, this test can be used to establish a relation between raw values and a defined power level: once such relation is established, is trivial to calculate the input power for other raw readings, provided a linear response of the circuit.

Algorithm 2.1 RTL Frequency response calibrator

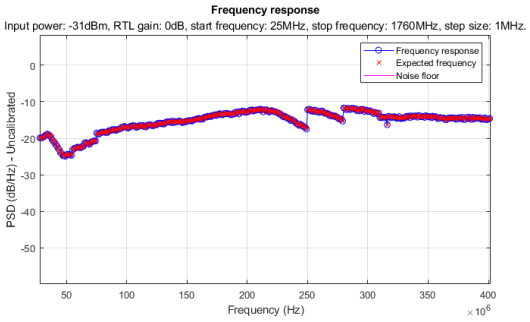
1: Open connection to SMA100B signal generator
2: Reset SMA100B to a clean known state
3: Configure SMA100B to output a CW signal
4: Configure SMA100B output power
5: Enable SMA100B output port
6: Connect to RTL-SDR with given parameters
7: Create vector of frequency to measure $frequencyVector$ spanning from $startFrequency$ to $endFrequency$ with step $resolution$
8: **for** $actualFrequency$ in $frequencyVector$ **do**
9: Tune RTL-SDR on current frequency $actualFrequency$
10: Configure SMA100B to output on frequency $actualFrequency$
11: Wait a short period of settling time
12: Capture three seconds of samples into $samplesVector$
13: $samplesVector = samplesVector - mean(samplesVector)$ {Remove DC bias from time domain signal}
14: Compute received signal power with Welch method with resolution of 1Hz
15: Find the power peak
16: Save the power peak raw value and its frequency in a vector
17: **end for**
18: Plot data
19: Save data to file for further analysis

As can be seen in algorithm 2.1, after the components and instruments initializations, the program enters a for loop that scans the list of frequencies chosen for the test. The spacing between the test frequencies was selected to be smaller than the bandwidth of the SDR, so that in each frequency the Nooelec can be tuned, at least two frequency points have been measured and can be used as calibration points, as it will be described in section 2.9. The results of the test are shown in figure 2.9. A power of -31dBm was used. The choice

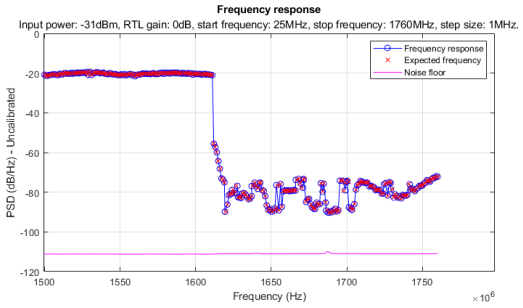
of this power level comes from the fact that being the frequency response not linear nor constant on the entire SDR spectrum, the highest power that does not saturate the ADC for a certain hardware configuration was used to obtain the best SNR possible.
In each figure, the noise floor is represented: this value was obtained averaging the power spectral densities of each frequency in the SDR receiving bandwidth after having removed the portion affected by the test tone. This figure helps visualizing the quality of the measurement, by giving a reference value about the presence of noise affecting it.



(a) Whole spectrum view.



(b) Dips at around 250MHz.



(c) Poor frequency response at around 1600MHz.

Figure 2.9: Nooelec SDR frequency response test results.

As can be seen in figure 2.9a, the frequency response is somewhat flat, with a slow decay as frequency rises, as expected. To quantify this attenuation, it can be said that the difference between the power measured at 200MHz and the one measured at 1600MHz is around 10dB. It can also be observed a slight oscillatory behaviour that's probably related

to resonances phenomenon with the used cable due to mismatches or input impedance mismatches, as it will be discussed in section 2.7. Anyway, the magnitude of this oscillation has been deemed negligible, as on average has an amplitude of 1dB. At the beginning and at the end of the frequency range, we can observe some strange behaviours: as visible in figure 2.9b, some dips and jumps in the frequency response graph can be clearly distinguished at around 250MHz. At a first glance, one may think we are observing some self-resonance of some component in the RF path. A closer look quickly led us rule against this hypothesis because of the sharp asymmetrical variations, uncommon in a self-resonance scenario. Most probably, what can be seen are jumps in the configuration of the RF filters in the tuner chip. To confirm this hypothesis, destructive testing on the hardware should be performed to directly check the input/ouput response of the tuner chip. As those jumps can be calibrated out in software, this testing phase was deemed unnecessary. At the end of the receiver spectrum an abrupt dip can be observed (figure 2.9c). Considering that the R820 tuner is designed to work under 1400MHz, the sharp attenuation at 1600MHz as to be considered not only something expected, but also an improvement of 200MHz on the declared limit. The noisy after this limit is not given by some sort of hardware or soft lock, but from instabilities within the R820 tuner chip above a certain frequency, as will be seen in section 2.6. To test the gain linearity, the same measurements where repeated with a power of -80dBm and a gain of 49.6dB, the maximum allowed one. This should lead to a signal to the ADC of comparable magnitude with the first one. Ideally, the slope of the spectrum curve should remain the same, indicating a linear and constant gain on the whole spectrum. As amplifiers have, in practice, a decreasing gain with increasing frequencies, some gain losses have to be expected.

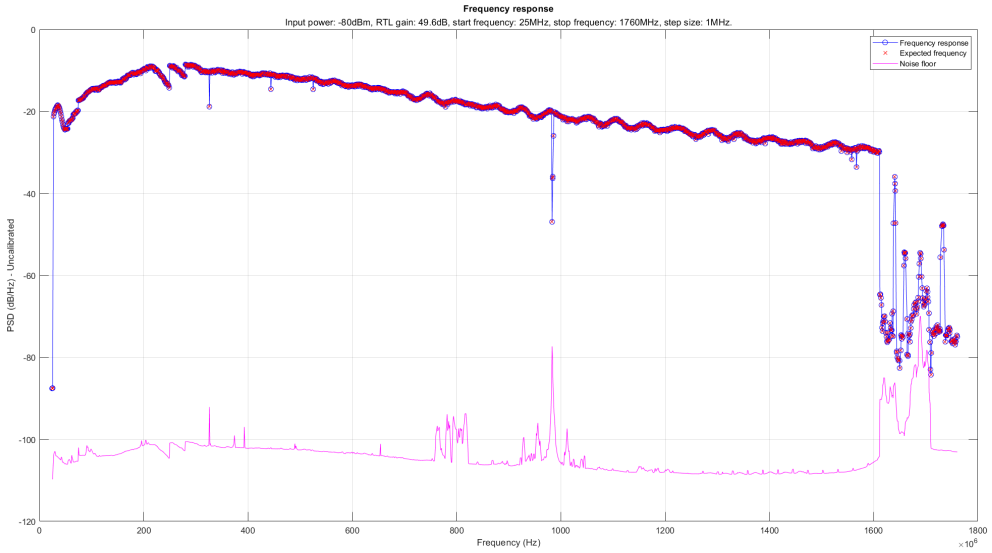


Figure 2.10: Nooelec SDR frequency response test results with -80dBm tone power.

Comparing figure 2.9 with figure 2.10, it can be observed how the measurements in the second figure are more strongly affected by noise, as the signal used has a lower SNR. Both the dips at 250MHz and the cut-off at 1600MHz are still present. What can be observed comparing figure 2.10 and figure 2.9a is the slope of the curve is steeper in the first case, indicating that the receiver gain is not linear with respect to the frequency. To both correct the response attenuation for a given gain and the gain non linearity, a 2D look-up table will be used in the final implementation.

2.6. Frequency stability

Nooelec specifies a temperature compensated crystal oscillator with precision of 0.5ppm. During the test described in section 2.5, for each test point was recorded the tone requested frequency and the detected tone frequency. In this way it was possible to measure the frequency error among the device frequency range. The results are shown in figure 2.11.

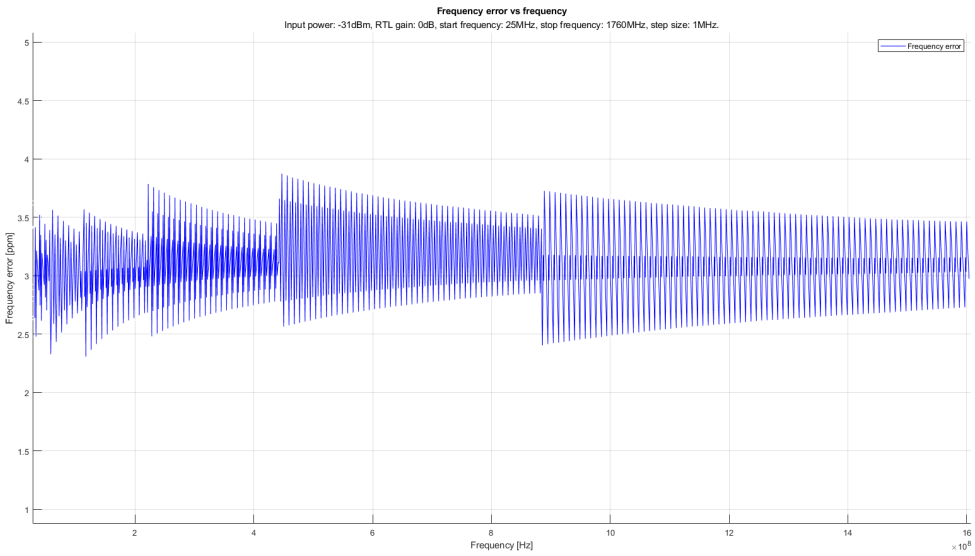


Figure 2.11: Nooelec SDR frequency error in ppm.

Despite what the Nooelec datasheet reported, a drift of around 3ppm (parts per million) was measured, that has to be attributed to the Nesdr, as the SMA100B has a clock stability of 10ppb (parts per billion)[22]. This result should not worry the reader: taking a closer look at the graph, it is possible to spot a pattern; the frequency error is probably mainly influenced by the discrete steps of the fractional PLL, meaning that an exact frequency can not be synthetized because a proper divider and multiplier are not available.
In figure 2.9c we saw how the response degrades and the device becomes unusable after 1600MHz. But is it really like that? During testing it was noted that the RTL device surface had the tendency to become very hot to the touch. The overheating of this SDR is a problem clearly known by the manufacturers, since the internals are stuffed with thermal pads to improve the cooling. To test if a better cooling really made any difference, the tests where repeated using a block of ice to remove the heat produced (test setup in figure 2.12).

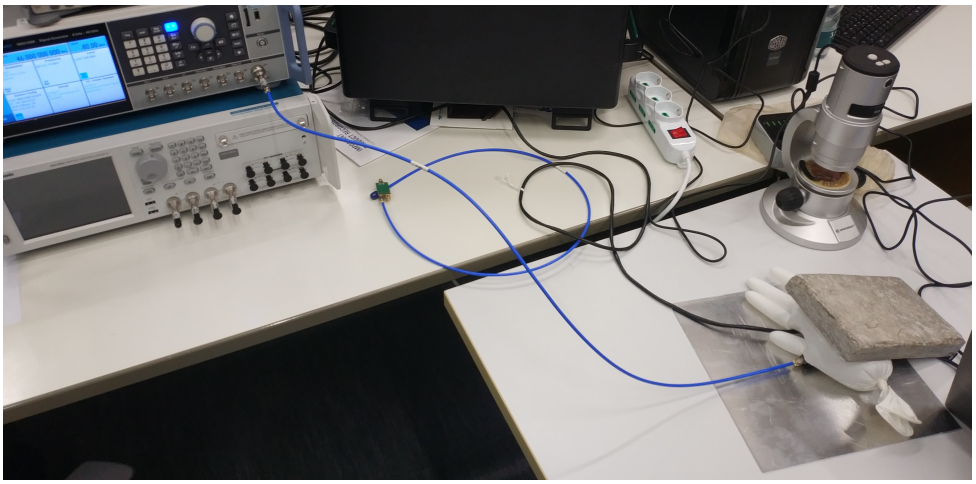


Figure 2.12: Cooled Nesdr setup: on the right the SDR (covered by the ice) is cooled by some ice kept in position with some weights, on the left the R&S SMA100B.

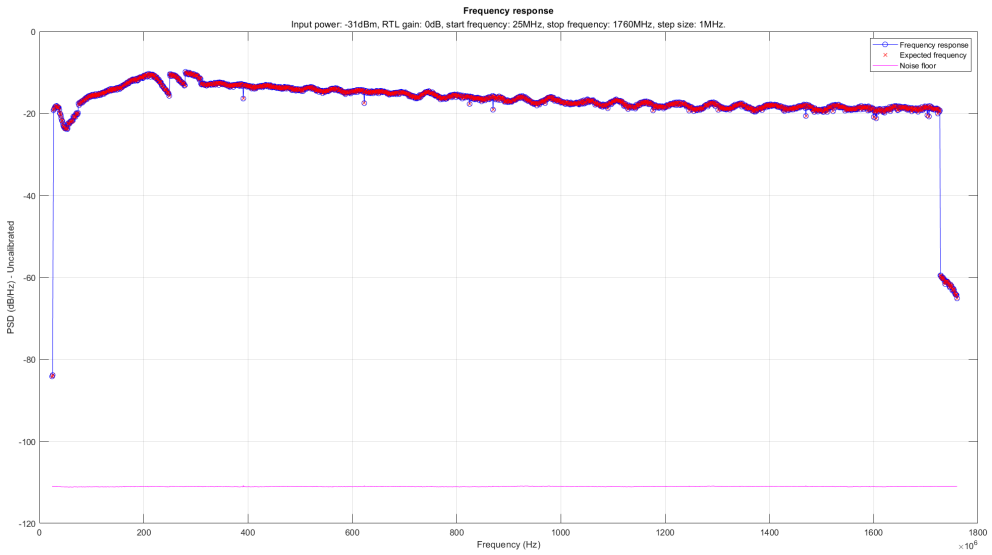


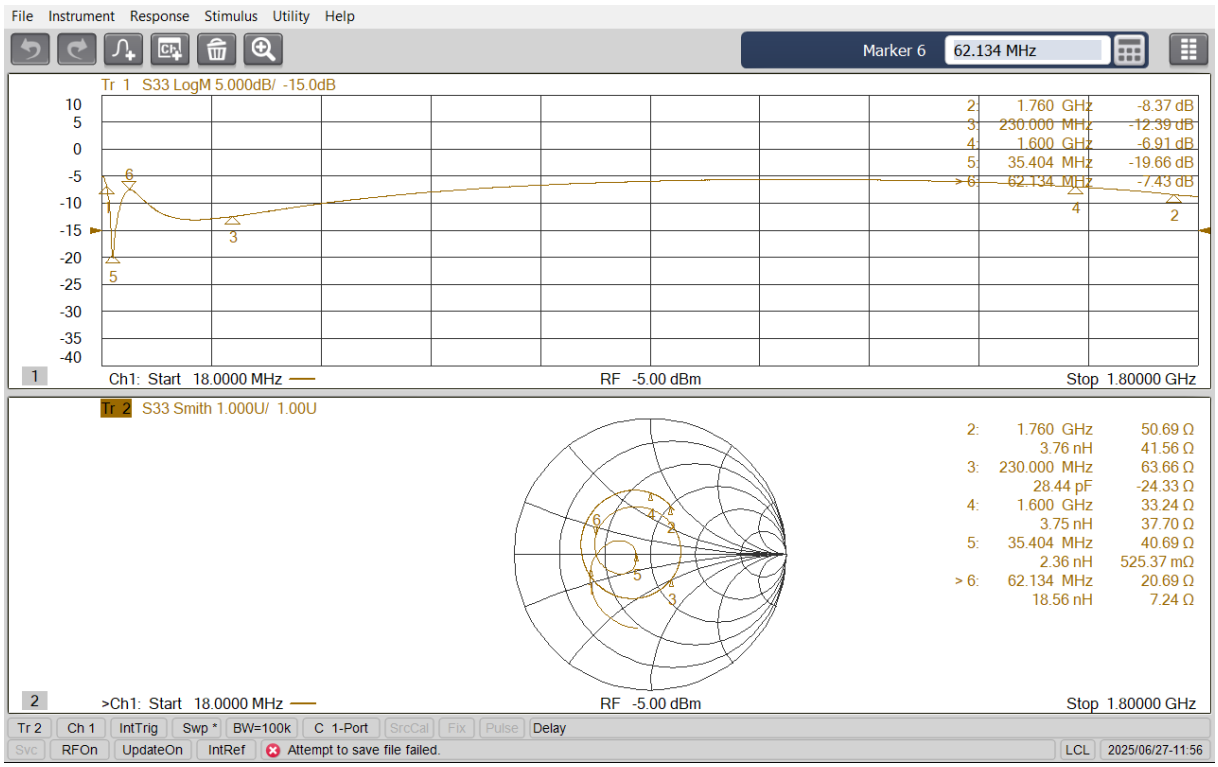
Figure 2.13: Cooled Nooelec SDR frequency response test results with -31dBm tone power. Improved performances compared to figure 2.9 in the portion above 1600MHz.

While the frequency error remained the same, a clear difference can be observed in figure 2.13. Most of the response curve remains the same, but it is possible to notice a great difference in the portion above 1600MHz. Here it is possible to discover that the SDR is not deaf at those frequencies, but a signal can be measured, albeit with strong attenuation. The abrupt jump is probably caused by an unsupported configuration of the tuner chip. On the other hand, after manual inspection, it was noted that the overheating of the SDR dongle caused the PLL inside the tuner chip to lose lock. This resulted in a device that

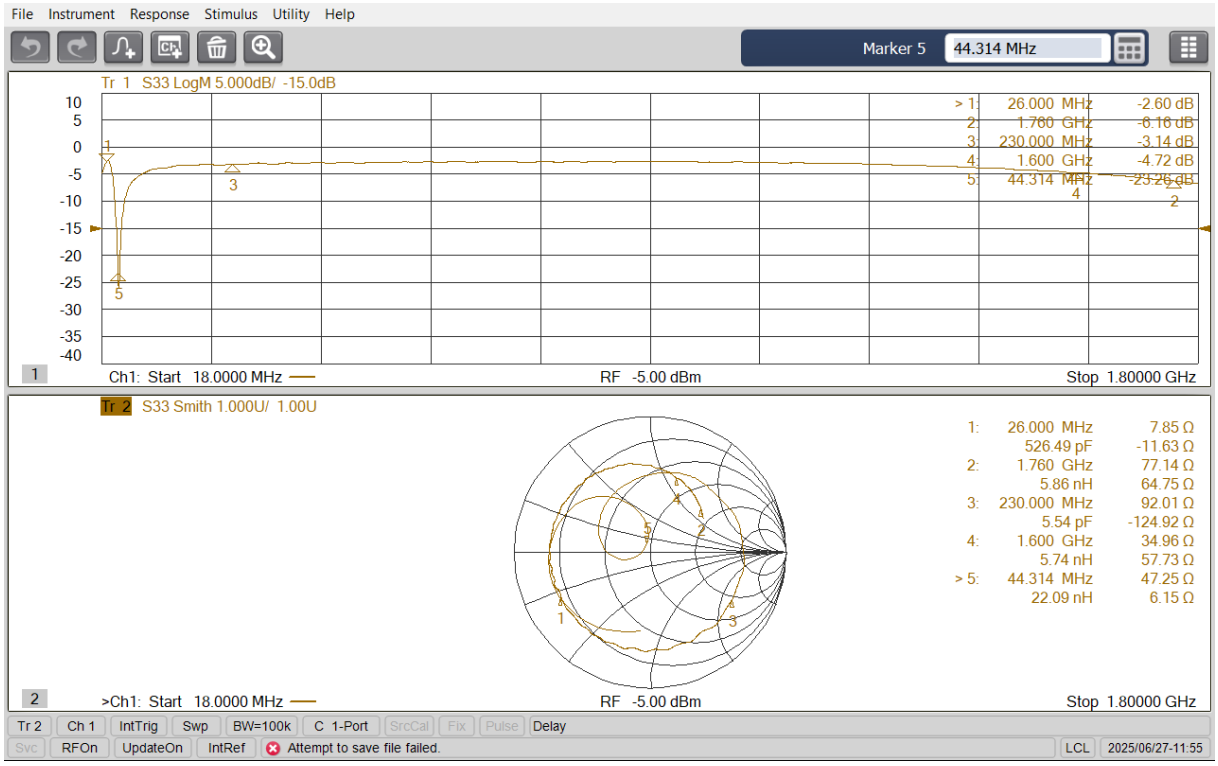
reported the correct tuning frequency while being tuned on another random one. This, in turns, mean that in figure 2.9c the portion of the frequency range above 1600MHz was not showing the actual attenuation of the SDR but some random noise. Being impractical to keep the Nesdr iced cooled in normal use, it was opted to restrict the maximum frequency range at 1600MHz, where stability problems still not emerged.

2.7. S11

To further explain the results observed in section 2.5 a vectorial analysis of the Nooelec SDR was performed using a vector network analyzer (VNA), more specifically the Keysight N5227B PNA[9]. Behing the Nooelec a one port device, the only measured parameter of the scattering matrix was the S11. The measurement was performed on a frequency range starting from 18MHz to 1800MHz. The S11 parameter is displayed on a Smith chart (figure 2.14,) to quickly check the input impedance, which the Nooelec datasheet says to be 50Ω[15], thus correct matching when coupled with other devices. Unfortunately it becomes apparent soon that it is not the case: the tuner input impedance is 75Ω and the input matching circuit (if present) is not able to correctly adapt this impedance to the 50Ω expected at the SDR input port. By plotting the S11 log-magnitude is possible to observe a behaviour resembling the one of figure 2.9: this suggest that the frequency response graph is not only affected by attenuation due to physical material properties of the components or filter combinations, but mostly by an heavy input impedance mismatch and return loss, that can be quantified between -5dB and -25dB.
To make thing worse, it was discovered that not only the input impedance is far from the declared 50Ω but, in addition, it unexpectedly changes when the tuner gain is changed in software as visible comparing the Smith charts in figure 2.14a, where the tuner gain is set to 0dB and in figure 2.14b, where the tuner gain is set to 49.6dB. Considering this results, the variability of the tuner impedance renders futile any attempt to design an impedance matching network.



(a) S11 measured with Nooelec tuner gain at 0dB.



(b) S11 measured with Nooelec tuner gain at 49.6dB.

Figure 2.14: Nooelec SDR vectorial analysis performed with a Keysight N5227B VNA.

2.8. IIP3

To have a complete understanding of the RF performances of the device, it was chosen to perform a two tones test to assess the IIP3, input third-order intercept point, which is a measure that gives an indication of the linearity of the device in term of input power response and intermodulation distortions. The idea behind the IIP3 is that a device non linearity can be modelled with a third-order polynomial. The third intercept point, a purely mathematical concept that usually lies beyond the irreversible damage threshold of a device, represents the point where the growth of the linear term and the growth of the third-order term intersect. Using two tones, the first at f_1, the second at f_2, device third-order non-linear term will appear in form of two additional tones at $(2f_2 - f_1)$ and $(2f_1 - f_2)$[10]. In can be demonstrated that, if far enough from the P_{1dB} compression point:

$$\begin{cases} IP_3 \simeq \dfrac{3P_{f_1} - P_{2f_2-f_1}}{2} \\ P_{f_1} = P_{f_2} \\ IIP_3 = IP_3 - G_{db} \end{cases} \tag{2.1}$$

Where G_{db} is the gain, IP_3 is the third order intercept point referred to the output port of the system, the ADC in our case, and IIP_3 is the third order intercept point referred to the input port, that is the RF port of the SDR. In this case, the figure of interest is the IIP_3, since the idea is to get a linearity figure in comparison to a given input power. The precision of the measurement is directly related to the purity of the tones used. Ideally, two CW generators like the SMA100B and a power combiner to sum the single tones would lead to the purest signal. As an alternative, an arbitrary signal generator, the Tektronix AWG70002B[30] was used. This device, with a 10 bits DAC resolution, capable of running up to 25Gsamples/s is able to generate an acceptable two tone signal, as shown in figure 2.16, albeit not as pure as using two single tone generators.

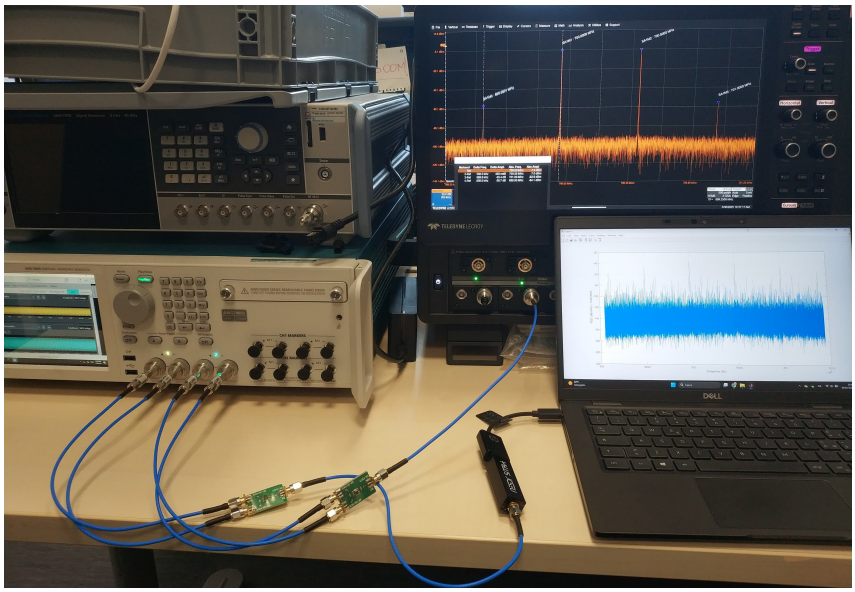


Figure 2.15: A picture of the IIP3 test setup. On the left, the Tektronix AWG70002B. On the right, the Nesdr connected to a computer. Behind, the Lecroy WaveMaster 8330HD MSO to check the used waveform.

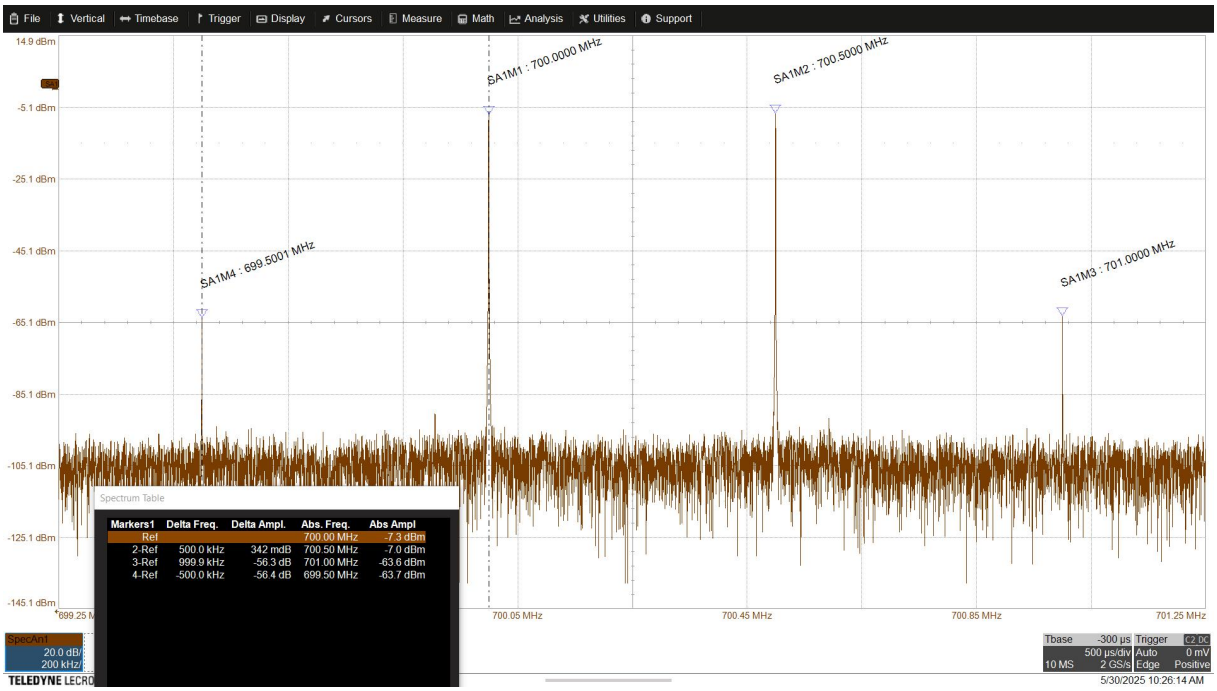
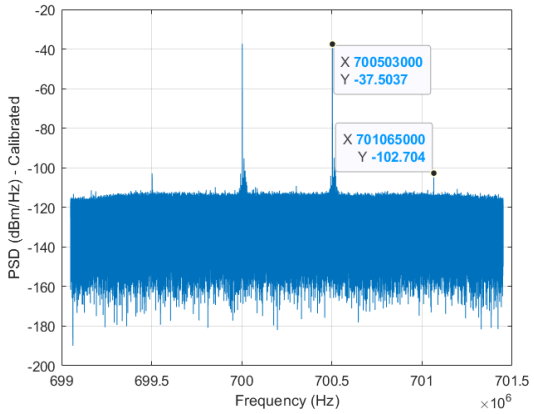
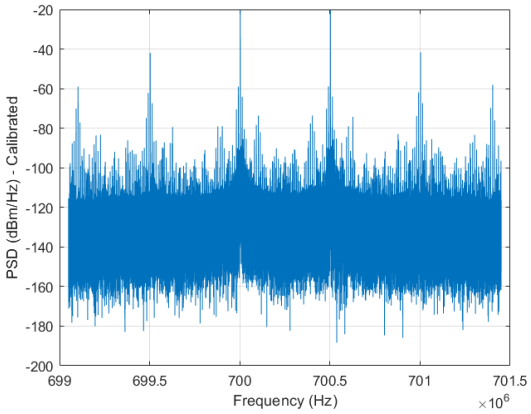


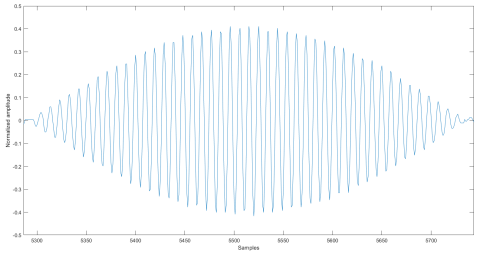
Figure 2.16: Tones at 700.0MHz and 700.5MHz and intermodulation products generated with AWG70002B. Signal acquired with Lecroy WaveMaster 8330HD MSO.



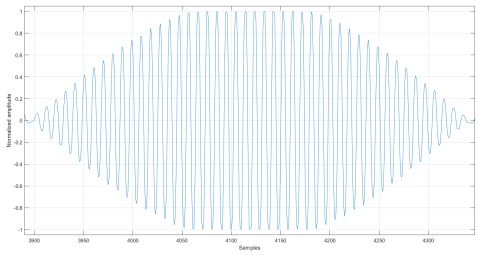
(a) Spectrogram with ADC not saturated.



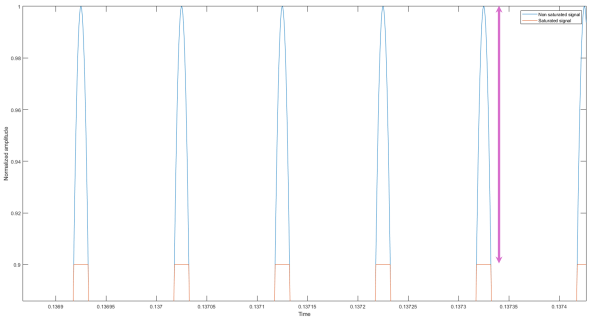
(b) Spectrogram with ADC saturated.



(c) Time domain capture with ADC not saturated.



(d) Time domain capture with ADC saturated.



(e) Comparison between saturated (orange) and non saturated (blue) signal

Figure 2.17: Nooelec SDR two tones test.

When acquiring the signal with the Nooelec Nesdr a problem arises: it is not possible to generate an appreciable intermodulation before the ADC saturates.
Looking at picture 2.17a is possible to see how the input signal is close to the full scale of the ADC, but still no appreciable intermodulation can be measured when comparing

to the reference signal of figure 2.16. Any further increase in the tones power leads to the situation of figure 2.17b, where the signal is destroyed by the ADC saturation. This is even more clear when looking the signal in time domain: while in figure 2.17c is possible to see the signal in its full dynamic, when reaching saturation, like in figure 2.17d, the portions with greater magnitude are getting clipped.
This result should not surprise: from a careful reading of the datasheet, it can be found that the $IIP3$ of the R820 tuner IC is 35dBm when tuner gain is set to zero, while ADC saturation is reached at -23dBm, a value too low to show any appreciable RF input path non-linearity.

2.9. Calibrated spectrogram

To generate a calibrated spectrogram, the program explained in section 2.6, and more specifically in algorithm 2.1 can be used. To precisely measure the received power, the Welch method is used instead of a standard DFT (discrete Fourier analysis).
After acquiring the signal for an arbitrary time (three seconds were used) the signal is saved in a vector. Any DC offset, that is usually present in a real measure, is removed in the time domain by simply computing the average of the signal and removing it from itself. Being interested in having a DFT with frequency resolution of 1Hz, considering the sampling rate and bandwidth of the signal of 2.4MHz, 2.4Mpoint must be used. The reader will have noticed that this value correspond to a second of signal, while three seconds were acquired. This is not an error: the Welch method used to obtain the power spectrum operates dividing the time domain signal in overlapping windows of, in this case, 2.4 million points. For each window the DFT is computed, then the results of each window are averaged. Doing this allows to reduce the variance in the measurement, obtaining more precise results.
If such measurement is repeated at equally spaced intervals over the whole SDR spectrum using a calibrated power test signal at a frequency equivalent at the one of each interval, a relation between the input power and the ADC readings can be established. This relation can be used as calibration factor:

$$CalFac = P_{dBm}tone\_used - P_{dB}read, \quad \text{if } RTLGain = 0 \tag{2.2}$$

that using linear units is equivalent to:

$$CalFac = \frac{P_{mw}tone\_used}{Pread}, \quad \text{if } RTLGain = 0 \tag{2.3}$$

The calibration factor represents, for each tested frequency, the difference in sensitivity of the SDR and can be used to calibrate the readings:

$$P_{dBm} = P_{dB} + CalFactor - RTLGain \tag{2.4}$$

Clearly, to have a meaningful calibration, the spacing between each test tone must be small enough to capture every rapid variation of sensitivity of the receiver, also depending on its frequency response curve flatness. In this project, two calibration tables were used, with 1MHz and 500kHz spacing: no major precision improvements were observed with the finer resolution. To obtain the calibration for frequencies between to calibration points, a linear interpolation can be performed.

3 | Signal generator device

3.1. Device description

As explained in chapter 1, a device capable of generating test tones is extremely useful, as one might want to perform specific measurements or assess the performance of an RF system using signals simulating a real use case.
With some limitations, an SDR with transmission capabilities can be used. The main limitation is dictated on how an SDR usually works, comparing it to an arbitrary waveform generator: while the latter has a fast DAC capable of generating the requested signal directly at passband, the first usually generates a baseband signal that is then shifted to the required passband frequency through an analog mixer. This imposes some constraints and limitations on the waveform that can be used and how the generation process and the output signal can be controlled, that are given by the fact that, while in the arbitrary waveform generator the output signal is actually what is generated by the DAC, that can be directly controlled in software, in the case of a radio transmitter there is not the possibility of digitally fine-tuning the output in a direct way, as the signal passes through the analog mixer.
Considering the modularity requirements described in chapter 0.1, the SDR device can be an half duplex transceiver, instead of full duplex. This because the almost totality of SDR on the market that are transmitters have also receivers capabilities, but, since we are only interested in transmission, the availability of the receiver functionality is not of interest. Indeed, while full duplex transceiver can transmit and receive at the same time, they usually tend to cost more. On the other hand, the reception and transmission functionalities in half duplex radios are mutually exclusive, but since receiver functionality will not be used on this radio, there will not be limitations in this use-case.
Because of the open-source nature of the project, the HackRF One SDR by Greatscott Gadgets[16] was selected. Also in this case, as it was for the RTL SDR (section 2.1), the device uses components designed for other purposes and adapted as generic transceiver. In particular, this SDR uses a MAX2837 monolithic transceiver chip from Maxim Integrated, originally designed for WiMAX transmissions[12].

Figure 3.1: Greatscott Gadgets HackRF One

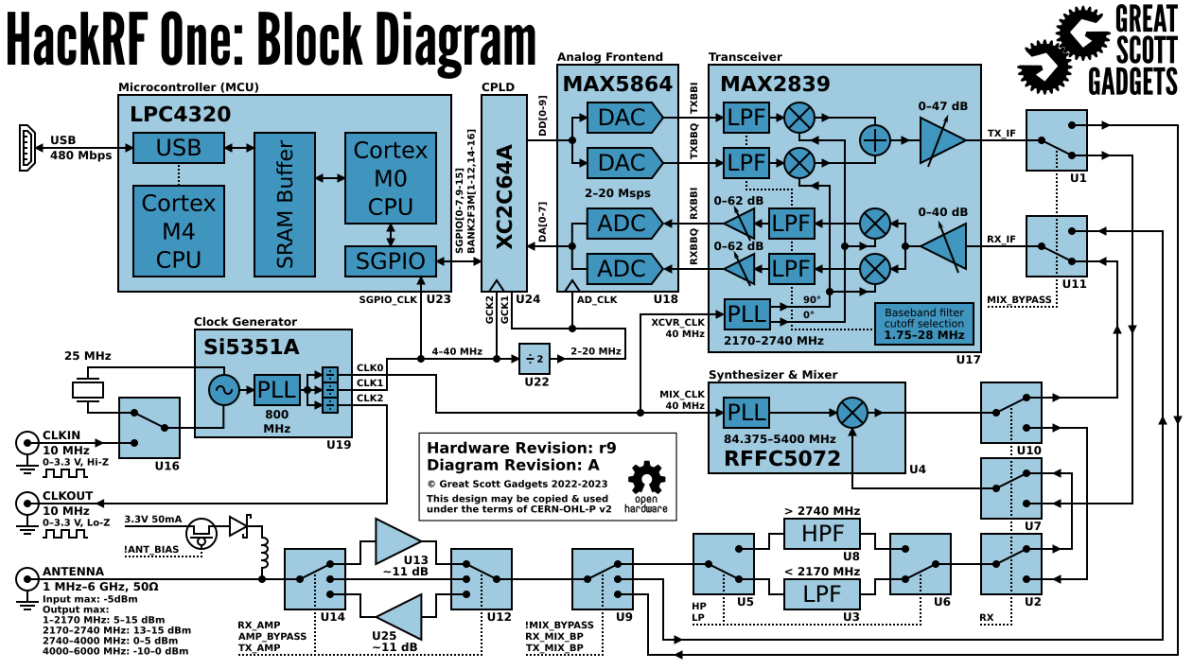
Although several versions of the Hackrf One can be found on the market, excluding revision 9, there are just minor differences among them dictated by the availability of components when released. The board revision 9 was designed to cope with the chip shortage in the COVID-19 pandemic time. Its major difference can be found in the MAX2839 replacing the MAX2837 and Si5351A replacing Si5351C. Both those chips have equivalent specifications and can be though as (except for the different pinout) drop-in replacements in terms of functionality. To be more specific, the MAX2839 is a MIMO transceiver chip that can be seen as two MAX2837 in parallel[13]: the design uses just one of the two transceiver modules. On the other hand the Si5351A, that is a clock generator by Skyworks Solutions, has less outputs compared to the Si5351C[26], but this does not pose a problem since many clock domains are just multiple others, so the same clock signal can be used with a proper additional divider (a comparison can be made with figure 3.3). The hardware is available from both from Greatscott Gadgets and from many vendors, mainly from China, since the projects adopts an open hardware license. While using secondary vendors can be considerably cheaper, great attention must be paid to the reputation of such vendors: some, in order to offer a better pricetag, use out-of-spec parts (both fake components or rejected ones from production lines), so that, while the SDR works apparently fine, an accurate analysis shows worse performances. This problem was encountered with one of the HackRF One used, resulting in an increase of intermodulation products and a greater local oscillator leakage, as will be descussed in section 3.6.
The HackRF can be controlled either with via USB connection to an host computer or through an add-on board called Portapack, becoming a standalone device. In this thesis usecase, the Portapack is not necessary, as a dedicated software (that will be discussed in chapter 4) running on a PC has been writed. Nevertheless, the Portapack is a great way

to display the device status: when controlling the HackRF via USB, the Portapack will display the current hardware configuration (figure 3.2).

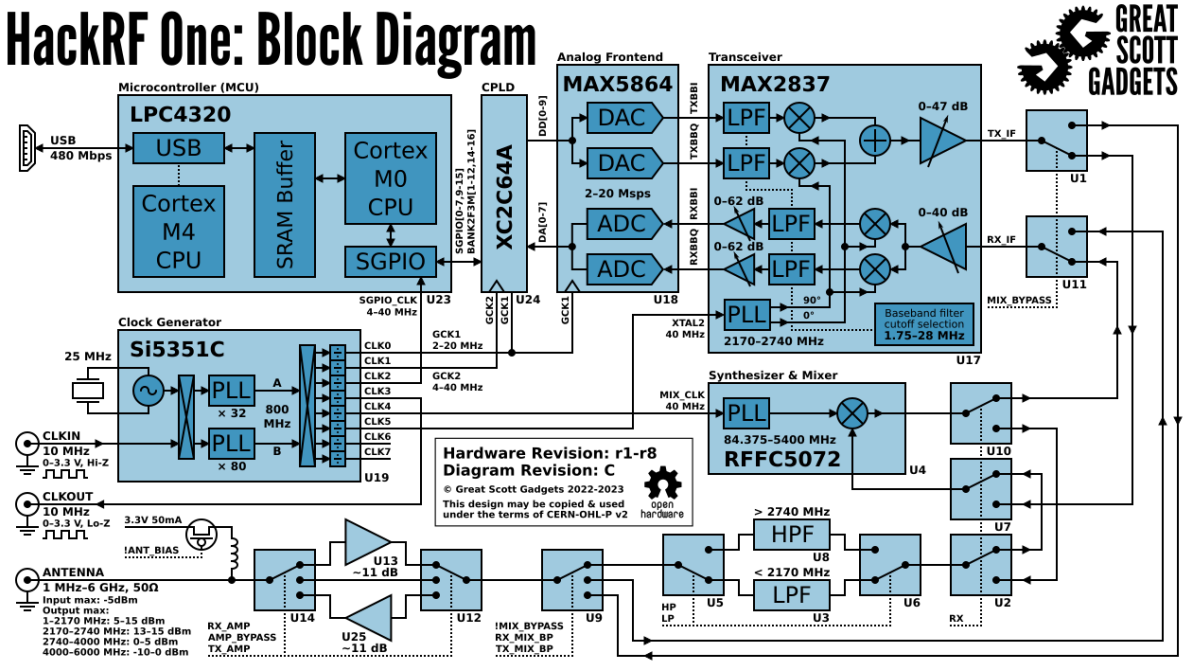
Figure 3.2: HackRF One with Portapack V2 transmitting into a dummy load.

3.2. Block diagrams

To better understand the inner working of the HackRF One, it can be useful to look at its block diagram. In figure 3.3 the complete block diagrams of HackRF are shown. Comparing the block diagram of revision 9 (figure 3.3a) with the one of revisions 1 to 8 and 10 (figure 3.3b), the differences mentioned in section 3.1 can be clearly seen: the Si5351A has 5 less outputs, anyway it does not represent a problem. In addition, it can be seen how the MAX2839 is functionally the same as the MAX2837 in this application. For this functional equivalence and similarities, the block diagram taken as reference in this paragraph will be the one in figure 3.3b.



(a) Revision 9.



(b) Revision 1 to 8 and 10.

Figure 3.3: HackRF One hardware block diagram.

The connection with the host computer is handled by the LPC4320 ARM microcontroller labelled U23 in the diagram. This component offers a USB interface and hold a FIFO

(first-in first-out) queue for received and to be transmitted samples. A complex programmable logic device (CPLD, U24) chip is used as glue logic to handle data transfers between the microcontroller and the DAC/ADC chip, the MAX5864. The MAX5864 is an integrated I/Q ADC and DAC, meaning that in the same package a matched couple of 8 bit analog-to-digital converters (one for the in-phase branch, the other for the quadrature branch) and a matched couple of 10 bit digital-to-analog converters can be found. Both of these couples can work to up to 20MHz, meaning that can handle a maximum of 20Msamples/s.
We will now focus on the transmission (TX) RF path, that is used in this application: being the device half duplex, the same path is used backward to receive external signals, so the signal follows the same processing in the inverse way.
The MAX2837 integrates a low pass filter with software selectable bandwidth on the baseband signal: this allows to easily remove many of the DAC generated spurious and harmonics out of the wanted signal. The filters are then followed by an I/Q modulator: a phase locked loop (PLL) inside the chip, which uses the clock generated by the Si5351 as reference, generate the intermediate frequency (IF) carrier at around 2.4GHz. This signal is used to drive two mixers together with the baseband signal and generate the IF/RF signal. Lastly, a variable gain amplifier can be used to boost the signal with a gain up to 47dB. At the exit of the MAX2837 several switches can be found: infact, the IF signal can be either up or down converted by an additional mixer, found in the RFFC5072 chip[18] or, if the desired signal lies in the portion of band around 2.4GHz, used without further processing. An additional filtering stage, composed by an high-pass and a low-pass filter is used to remove the image frequency generated with the RFFC5072 mixer, when used. Lastly, the signal can be further amplified with an additional fixed gain amplifier by about 11dB[1].
From the block diagram and what previously said, it can be easily understood that the HackRF One can operate on a range much wider than the Nooelec Nesdr: indeed, while the Nesdr can operate on frequencies from 26MHz up to 1700MHz, the HackRF can operate from 1MHz up to 6000MHz. Technically, there are no limitation in our application to use the HackRF on its full range as a standalone signal generator. Wanting to avoid confusion in an inexperienced user, favouring the use in conjunction with the Nesdr, a software lock in the companion app (discussed in chapter 4) on the frequency range has been put in place, limiting it to the one of the Nesdr. This limit can be easily removed by changing one line of code if a user desires to do so. Considering this and the specific application of this project, the block diagram can be simplified as the one in figure 3.4, where the unused parts are removed.

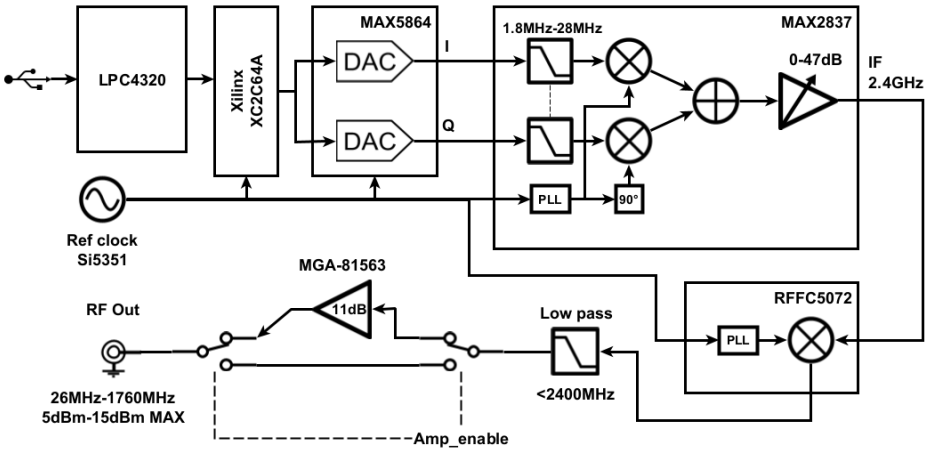


Figure 3.4: HackRF One simplified TX path used in this project.

3.3. Drivers

A driver, called libhackrf, written in C++ is provided within the HackRF project. The driver uses libusb to handle the USB communications stack.
Unfortunately, due to how the drivers are implemented, despite the DAC having 10 bits resolution, only 8 bits can be used, as the only data format supported is of type int8_t, meaning 8 bit integers. This leads to quantization errors passing from high resolution signals in software to an approximation to 256 levels in hardware, with the generation of unwanted spurious.
The HackRF's LPC4320 microcontrollers holds a FIFO (first in, first out) fixed length buffer. The I/Q samples are transferred in blocks to limit USB transfer overheads, interleaving I and Q 8 bit data. The library is released as source code to be compiled on the host machine. It has been observed that the compatibility of the compiled library is not dependent on the single machine hardware, but on the combination of operating system and hardware architecture. This means that an X64 PC running Windows 11, which are the majority of moderns PC, can rely on the already compiled version by the author, without the need to install the whole complex compiling chain.
An official MATLAB interface is not available. An old and no longer maintained library has been found on MATLAB File Exchange [28]. Unfortunately, the lack of documentation posed a challenge in implementing the interface. Moreover, the interface presented some bugs and errors. For those reasons, the interface was rewritten and published with a new support manual. The problems found related on handling of datatypes and consistency between hardware status and software representation:

- The function that handles the output amplifier was not working as a mismatch between MATLAB datatypes and C datatypes was present, causing errors.
- MATLAB automatically approximate complex numbers with null imaginary part to a real number variable type. This, if unchecked leads to a catastrophic crash caused by a mismatch between the expected data length by hackrflib and actual received data, leading to a buffer overflow. The problem has been fixed enforcing complex number datatype.
- Some commands, like the one to set the VGA gain of the MAX2837 simply ignored values outside the allowed ones, without throwing any warning. The behaviour was fixed adding a warning and automatically rounding the requested gain value to the closest available one.
- It has been added a method to purge the HackRF internal buffer.
- The output amplifier is automatically disabled by the HackRF firmware when it passes from a transmitting state to an idle state. This was not reflected in software, creating a coherence issue. With the actual library implementation, when a HackRF exits the idle state, its previous configuration is restored.
- Warnings and errors messages have been added to inform users of wrong usage of the library before an irrecoverable crash can occur.

The library is composed by two main parts: a .MEX file, that is a MATLAB package to access C libraries[32], acting as a shim, and a MATLAB code to be used by users as interface. The interface is implemented as an "HackRF" class, where with "interface" it is intended an API (application program interface). A HackRF software object is created at runtime representing a physical HackRF. The object presents specific methods to set and get specific hardware parameters. Each method works by invoking a specific hackrfdev, which is the .MEX file, methods, which in turn call different hackrflib routine to correctly configure the hardware, as illustrated in figure 3.5. The transmission process is implemented using a callback routine: a callback function handler is passed to the HackRF software object; when the hackrf_start_TX method is invoked, a timer that periodically calls the provided callback function is started. Each time the callback function is called, a number indicating how many samples can be generated to fill the physical HackRF buffer is passed as an argument. The function returns the requested number of samples that are then passed to the physical device. This implementation allows the programmer to define a signal generating function without the need to worry about timing issues and data transfers, as these problems are handled by the interface logic. The HackRF physical

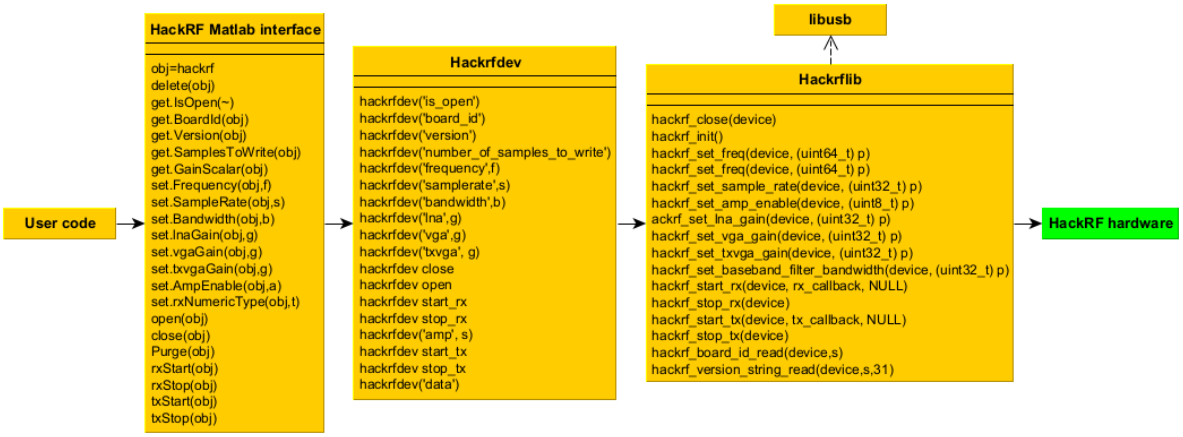


Figure 3.5: HackRF MATLAB interface logic.

hardware stores those samples in its internal buffer and transmit them at a constant sample rate defined with the interface command "hackrf_samplerate".

3.4. Bandwidth & sample rate

Ideally, the digital signal sent to the HackRF to be transmitted should have a sample rate high enough to correctly represents every change in the waveform. This means not only satisfy the Nyquist sampling theorem, but also account for the specific characteristics of the hardware used: since no interpolation is made by the HackRF, the provided signal should also have a sample rate high enough so that there are enough samples to smoothly represents signal transitions, avoiding the generation of unwanted spurious with abrupt signal changes, that might affect the quality of the output signal. Being the maximum bandwidth of the receiver SDR is just 2.4MHz, ones could think that a sample rate of 4Msamples/s (considering complex samples) might be enough. Unfortunately, while the HackRF would work with such sample rate, the MAX5864 DAC would work outside its specifications: the recommended sample rate for this device by the manufacturer is indeed between 8Msamples/s and 20Msamples/s[11].
While Greatscott Gadgets markets the HackRF capable of working up to 20Msamples/s, for an equivalent bandwidth of 20MHz, during experimental testing the maximum fastest sampling rate that offered a stable operation was 12.5Msamples/s[1], due to the used configuration. Above this rate, underflow event started occurring, becoming more frequent every time the sampling rate was further increased. The probable cause for this behaviour has to be found in the language used: MATLAB is both an interpreted and compiled lan-

[1]The HackRF can in theory use any sample rate frequency between 2MHz and 20MHz, but some predefined values are recommended: 2.4Ms/s, 4Ms/s, 8Ms/s, 10Ms/s, 12.5Ms/s, 16Ms/s, 20Ms/s.

guage adopting a "just-in-time" compiler, that means that in many context the code is interpreted line by line or blocks are compiled on the flight during runtime. While this allows for great flexibility and easy debugging, it also adds considerable overheads, meaning that the program execution can not keep up with USB data transfer and the HackRF buffer gets empty before the program has the opportunity to send new samples, generating underflow events.
Luckily, this does not pose a problem in this project, as the signals used are designed to have a maximum bandwidth of 2MHz. An oversampling factor between 3 and 6, equivalent to a sample rate of 8Msamples/s to 12Msamples/s has proven to be more than sufficient to fully utilize the DAC capabilities, pushing most of the spurious in the stop region of the baseband filters discussed in section 3.5.

3.5. Baseband filtering

The MAX2837 provides a couple of powerful low-pass filters to improve the quality of the baseband signal. While those filters cannot remove in-band spurious, they are effective in removing harmonics and some quantization noise and spurs. The filters bandwidth can be selected among 16 discrete steps ranging from 1.75MHz to 28MHz. Since the maximum theoretical sampling rate o the DAC is 20Msamples/s, filters with bandwidth above 20MHz are of limited use. Accounting for this, still 14 bandwidth combinations are available. To obtain an experimental measurement of the filter frequency response a test was designed. A MATLAB script generating a wideband white noise signal was written. The generated signal has the characteristic of having uniformly distributed power spectral density (PSD), as expected for white noise, on a large bandwidth, much larger than the passband filter ones. The script sets the MAX2837 baseband filter to the requested bandwidth to be tested. For this application, the focus was posed on the 1.75MHz bandwidth, as it is the closest to the Nooelec acceptable bandwidth. The MAX2837 effectively shapes the constant power signal attenuating the out-of-band portions before mixing the baseband signal with a carrier and outputting it. If a carrier frequency between 2.3GHz and 2.7GHz is selected, the output signal is passed by the HackRF directly to the RF port (figure 3.3b), bypassing any additional component that could add distortion. Exploiting the zero mean statistical property of the white noise after feeding the signal to a spectrum analyser and averaging its readings, it is possible to obtain a clear trace of the filter frequency response. In figure 3.6 it is possible to see the response curve of the 1.75MHz filter. As the drivers do not allow to disable the lowpass filter, an insertion loss value can not be provided. What instead is appreciable is the flatness of the filter in the passband region. The frequency response of the stopband region and its eventual ripples are not

appreciable, as the attenuation in this frequencies pushes the signal at a level lower than the noise floor. The bandwidth of the filter has been verified to be not exactly 1.75MHz, but 1.56MHz, if considered in a baseband scenario. The reader must not be fooled by the shape of the filter in figure 3.6: being the filtering carried out in baseband, before the modulation and upconversion, the expected frequency envelope of the signal after this process, at passband, is infact the one of the baseband case mirrored at around the carrier frequency. The difference between the measured filter bandwidth and the specified one can be explained keeping in mind the target application of the MAX2837 IC: being a WiMax modulator. This plays an important role when comparing the filters bandwidths with WiMax specifications[38]: the filter were intended to satisfy the bandwidth requirements of specific WiMax modulated signal and not as generic lowpass filter. Continuing using as an example the 1.75MHz filter, in order to meet WiMax specifications, its frequency response should be flat out to 875kHz at baseband, such that there's an at-least 1.75MHz wide flat passband at RF. Considering this, it is now apparent why the precise filter bandwidth was not the primary requirement in chip manufacturing. To mitigate this problem, the IC offers a way to digitally tune the filter passband up to a value of ±10%. Unfortunately, this operation has to be performed in the firmware and there is no easy way to do so on the HackRF.

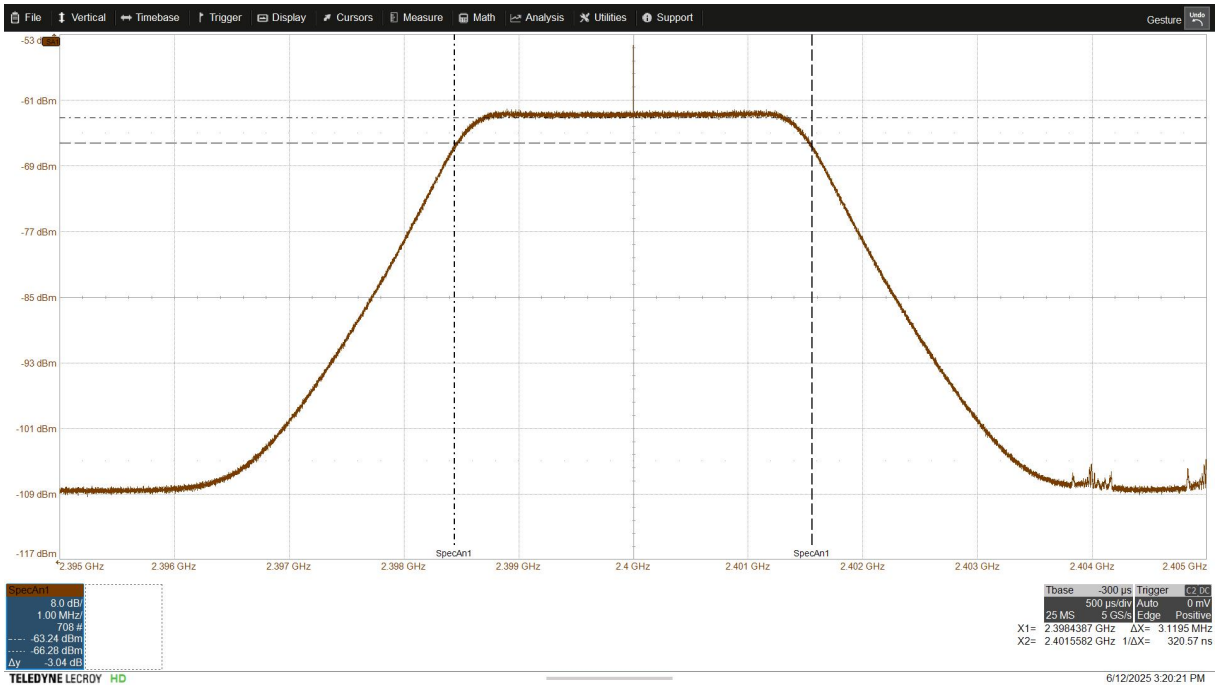


Figure 3.6: HackRF One 1.75MHz bandwidth filter response. Signal acquired with Lecroy WaveMaster 8330HD MSO.

3.6. Non-linearities and spurs

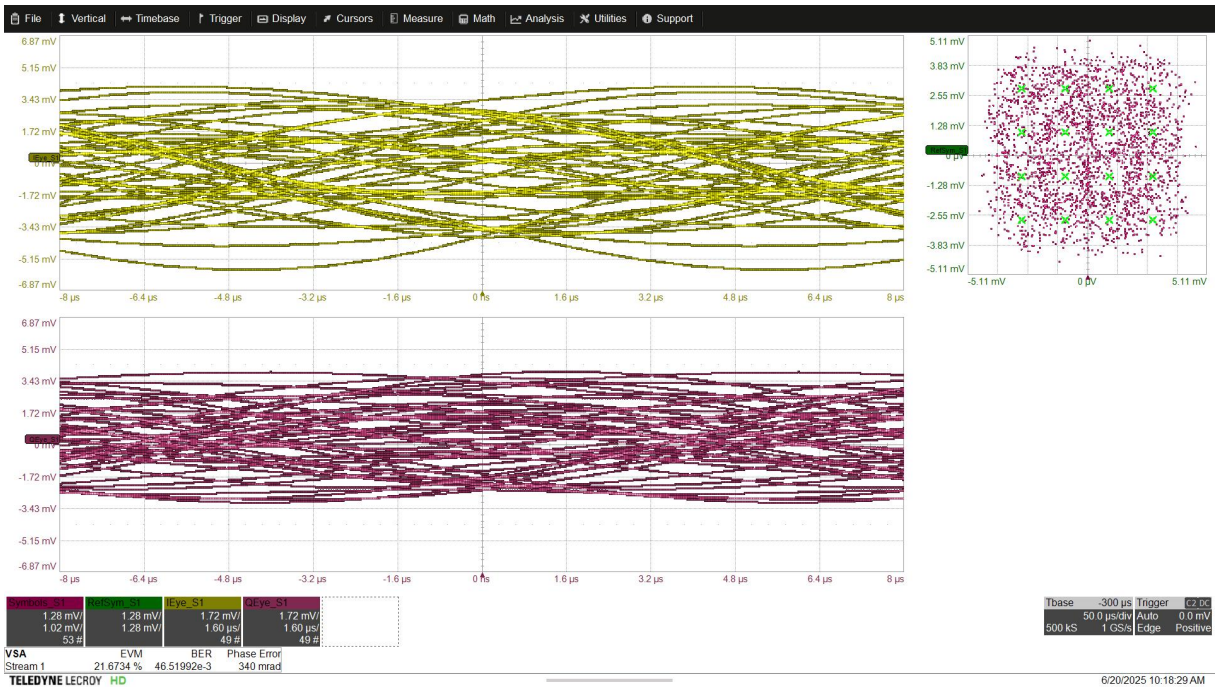
Using patterns (that will be discussed in depth in chapter 4.4) on two different HackRFs with the same settings, it was noted a great difference in the two output signals. It was repeated the filter characterization test of section 3.5 on the two different devices, the first one, that will be called "HackRF 1", was bought during the COVID19 pandemic, the second one, "HackRF 2", was bought during this work development, both from different Chinese manufacturers. The results are visible in figure 3.7: the orange trace represents the first SDR (HackRF 1), while the green trace represents the second newer SDR (HackRF 2). While the bandwidth of the filters is about the same, a considerable difference of the insertion loss value of about 10dB can be seen. Moreover, an heavy local oscillator leakage, affected by a serious spurs content, can be spotted in the HackRF 1 signal.

Figure 3.7: HackRF comparison: good performing device - HackRF 2 (green trace) and faulty device - HackRF 1(orange trace) 1.75MHz filter response. Signal acquired with Lecroy WaveMaster 8330HD MSO.

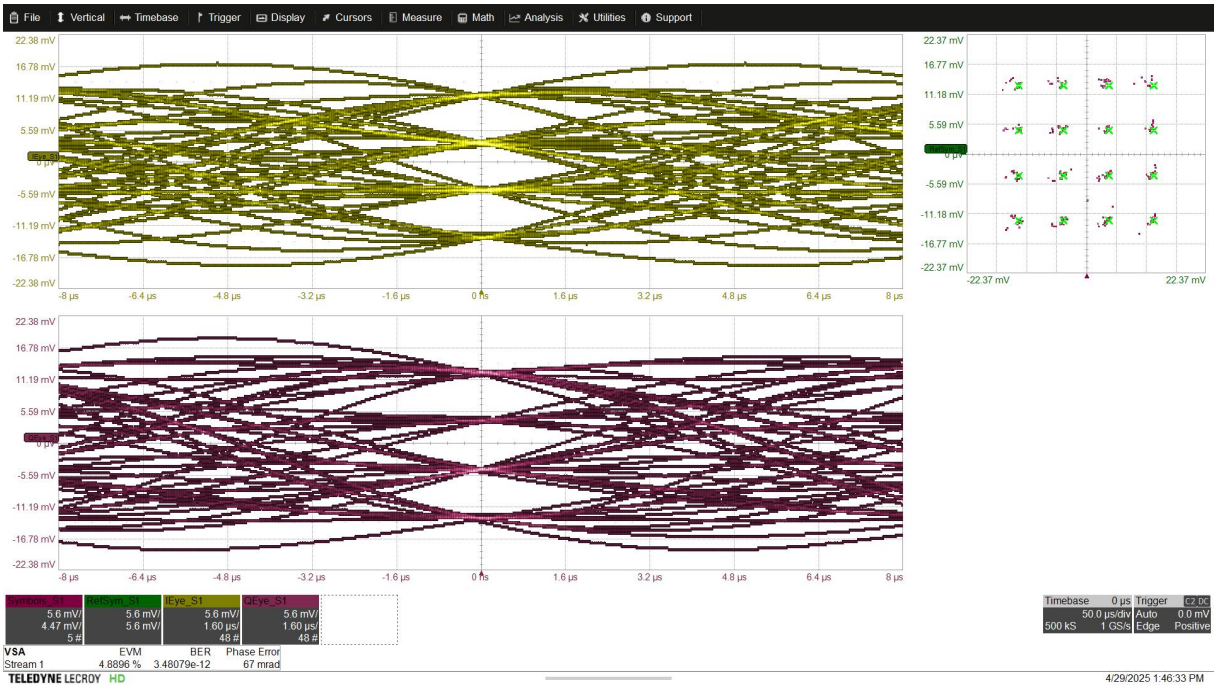
A suggestion of what could be happening can be guessed by thinking at the manufacturing date of the first device: a period where a severe chip shortage was present. Upon closer inspection, after tearing down the two SDR and comparing the two PCBs (printed circuit boards), the hypothesis can be confirmed: the markings of many key chip of

the first HackRF, such as the Xilinx CPLD and MAX2837 modulator, appear to have slightly different graphics and fonts comparing with what it is illustrated in their respective datasheets. This leads to the conclusion that, in order to cope with the lack of genuine chips, fake ones (either from a different manufacturer or not passing quality check on the manufacturing lines) were used. Sadly, after reviewing the HackRF support forum, it appears that this is a quite common problem, not only caused by the pandemic crisis, but still actual. To make thing worse, it is quite difficult to discover fake chips by just looking at them and almost impossible to find it out at the moment of purchase. From a signal quality prospective, the first SDR is almost unusable: tones are affected by a strong spurious content, while digital signals are affected by strong random I/Q impairments, in form of varying random DC offset and I/Q branches imbalance. A clear comparison can be made in figure 3.8: in the first image, representing the analysis of a QAM (quadrature amplitude modulation) signal generated by the first SDR (figure 3.8a), the eye diagram is closed and noisy. The constellation diagram shows symbols scattered over the plane and both the EVM (error vector magnitude) and BER (bit error rate) are very high, indicating a poor quality signal. On the opposite, figure 3.8b it's possible to observe the signal analysis of the second good performing SDR (Hackrf 2), using the same signal and configuration of the first one: the eye diagram is open, with clear nodes; the constellation diagram shows symbols well positioned respect to reference ones. This is confirmed also by a low EVM of around 4% and a low BER.

Several patterns were tested to assess the response of the SDR. One of the most indicative, also considering the final target application, is a signal with two tones. The generated signal was measured with an oscilloscope and the results are shown in figure 3.9. Two settings were tested: in figure 3.9a it is visible the spectrum of the signal generated using the baseband filter discussed in section 3.5 with a setting of 1.75MHz, while in the second run, visible in figure 3.9b, the filter bandwidth was set at 10MHz to compare the effects of filtering the baseband signal against using a filter so wide that it does not have appreciable effects (as previously mentioned, filters cannot be disabled). It can be seen how the desired tones, at 100.5MHz an 99.5MHz, have comparable power among the two cases. What dramatically change are the harmonics and intermodulation products magnitude, proving that a narrow filter does actually makes a difference on the signal quality. The origin of those unwanted signal components can be traced to various sources: part of them are generated in the MAX2837 modulator as intermodulation products; others are generated by the second stage mixer, the RFFC5072, that translates the IF frequency to the required RF frequency, and starts saturating when the MAX2837's VGA has a gain of more than 38dB; most are generated by the DAC itself, as proven by filtering the DAC signal at its output with the baseband filters. This DAC generated spurious are

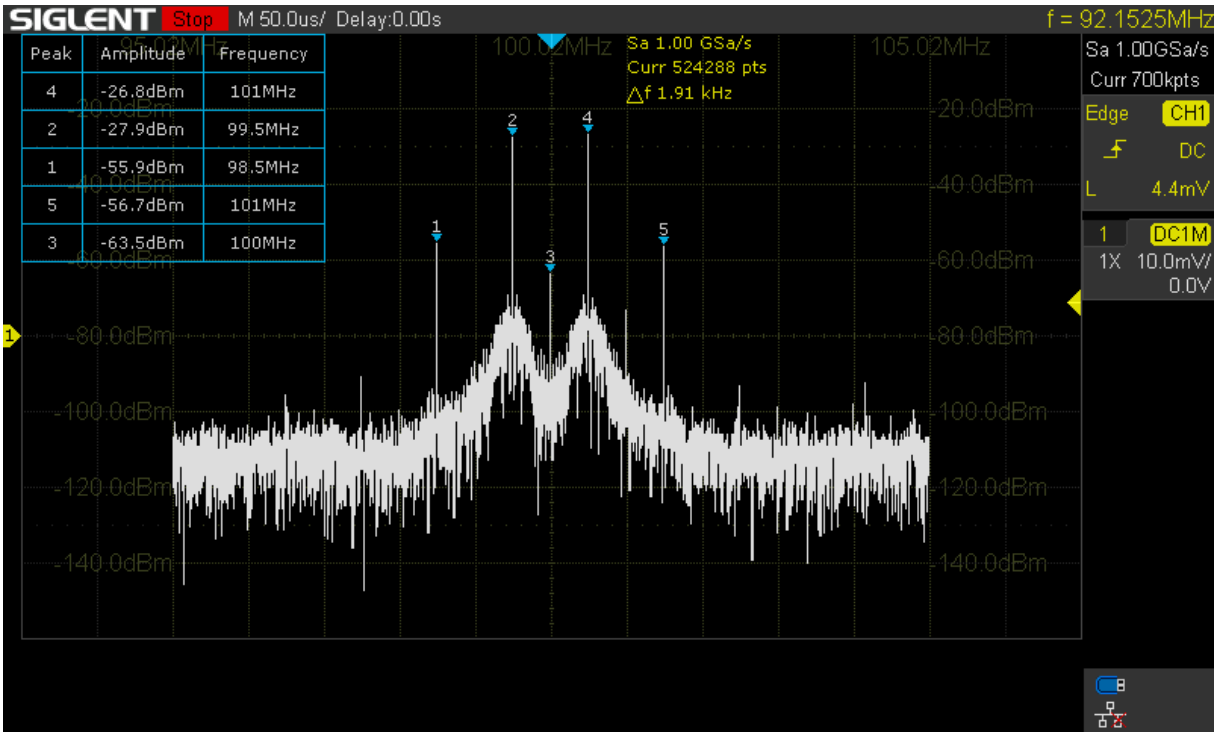


(a) QAM signal from HackRF 1.

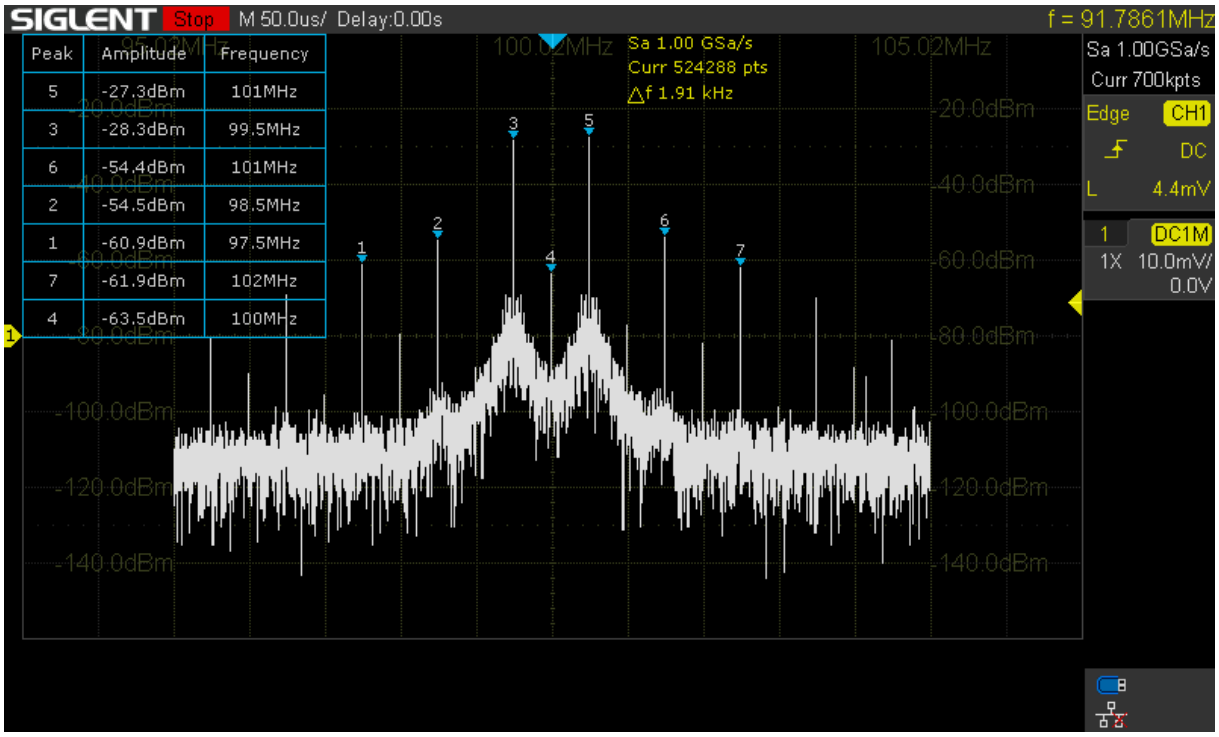


(b) QAM signal from HackRF 2.

Figure 3.8: HackRF QAM signal analysis. Signal acquired with Lecroy WaveMaster 8330HD MSO.



(a) Two tones signal, 1.75MHz baseband filter.



(b) Two tones signal, 10MHz baseband filter.

Figure 3.9: HackRF two tones signal, 24dB gain, 12.5Msamples/s. Signal acquired with a Siglent SDS1204X-E oscilloscope.

probably mainly due to the HackRF design not fully utilizing the full DAC range, using only 8 bits instead of 10 bits, as mentioned in section 3.3. This leads to a rounding of the actual sample's value, causing jumps in the waveform, defined as quantization errors, thus spurious.

Another measurable signal defect is a local oscillator leakage from the mixer and baseband DC component, present as a small tone (marker 3 in figure 3.9a) in the middle of the spectrum. Usually, this tone remains below 40dB from the desired signal. When a signal with higher purity is required, it is difficult to remove this tone because of its position inside the spectrum. For this reason a solution has been proposed: as visible in figure 3.10, the signal generated is not anymore centered around the carrier frequency. Instead, one of the two tones lies directly onto the carrier frequency, thus onto the signal leaked by the mixer, effectively masking it. This was made possible using an Hilbert transform on the desired signal[2], as will be explained in chapter 4.4. The main disadvantage of this technique is that only half of the filter is now used and the maximum tone spacing for every given filter bandwidth is reduced. Moreover, the intermodulation products on the left of the signal have higher magnitude (marker 1 and 2 in figure 3.10) as they lie in the passband zone of the filter. On the other hand, it becomes easier to use an external passband filter to further clean the signal and achieve a higher purity.

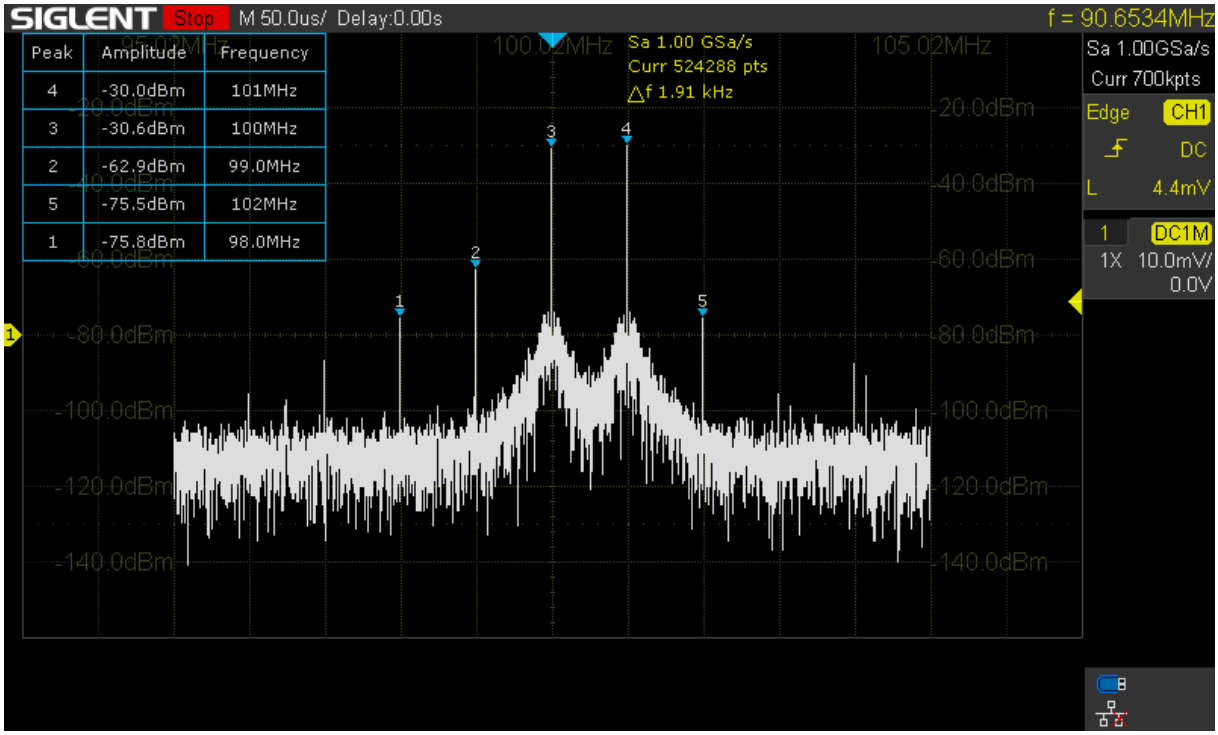


Figure 3.10: HackRF two tone signal shifted using Hilbert transform. Signal acquired with a Siglent SDS1204X-E oscilloscope.

3.7. Power output control

It is difficult to give a useful maximum power output value and controlling it through software. This because the maximum power depends on the transmitted signal bandwidth, center frequency and signal purity requirements. As an example, while a single tone can be generated with 0dBm power, this is not the case for a two tones signal and even worse for digital signals. Why is it so? There are two main reasons: one related to the physical operation of a radio system, and the other related to signal quality considerations. In the following reasoning, it will be assumed that the full dynamic range of the DAC is used, so the baseband signal will always be generated with the maximum amplitude possible to maintain resolution, thus dynamic range. Let's discuss the first reason following a practical reasoning: being the HackRF a real hardware, its available output power is fixed, meaning that, at any given time, at the HackRF output port, independently of the signal used, there is a power level that cannot be physically exceeded. From a frequency domain prospective, this means that when using a single tone signal all the available power is carried by the single frequency component present in the signal, while, for example, in a 1MHz QAM (quadrature amplitude modulation) signal the available power gets spread over the 1MHz band. From a formal point of view the concept of power spectral density (PSD) can be taken into consideration: the total power of a signal is given by the contribution of each frequency component, so at each frequency can be associated a weight, the PSD, that indicates how much the specific frequency contributes to the total power of the signal. Then, the power of a signal can be found with:

$$Power = \int_{bandwidth} PSD(f)\, df \tag{3.1}$$

From equation 3.1 seems trivial to obtain the total output power, but unfortunately is not the case: one must define a power distribution function for each arbitrary signal used, taking also into account the frequency linearity dependence of the available power output. More importantly, one might be interested in setting a predetermined target power output that the signal must have. Also in this case, some complications arise: thinking about non constant amplitude signals, so amplitude modulated ones, without constant power, how should the target power be intended? Most of these problems are solved on most systems using specific hardware: a power detector placed at the RF output of the systems gives an instant power reading independent of the signal used; these readings can eventually be used in a software control loop. Unfortunately the HackRF has not such capability. For these reasons the power control settings has been tied to the variable amplifier gain settings: this means that the user cannot set a specific power target, but set an amplifier

configuration to reach a desired signal output amplitude. Considering that the HackRF is thought to be operated in tandem with the Nesdr in this application, it becomes trivial to check the effective characteristics of the generated signal by simply measuring it directly with the Nooelec Nesdr, while tweaking the settings. Once the desired settings have been found, the user can operate the system confidently knowing the expected generated signal (this idea will be exploited in section 4.9).
Other than the IF VGA (see figure 3.3), an RF amplifier is available to further increase the generated signal by 14dB. Attention should be paid when using this amplifier: setting the IF VGA above 30dB gain is enough to push the output amplifier in a non-linear region, where intermodulation products start to become apparent. Keeping in mind this consideration, the second reason mentioned at the beginning of this section becomes now clear: pushing the HackRF gain at its higher limits greatly degrades the signal quality. While this can be acceptable when using simple tones, as most of the spurious and intermodulation products are in an out-of-band region, this is not true anymore when talking about digital modulated signals, where in-band distortion starts to become impactful.

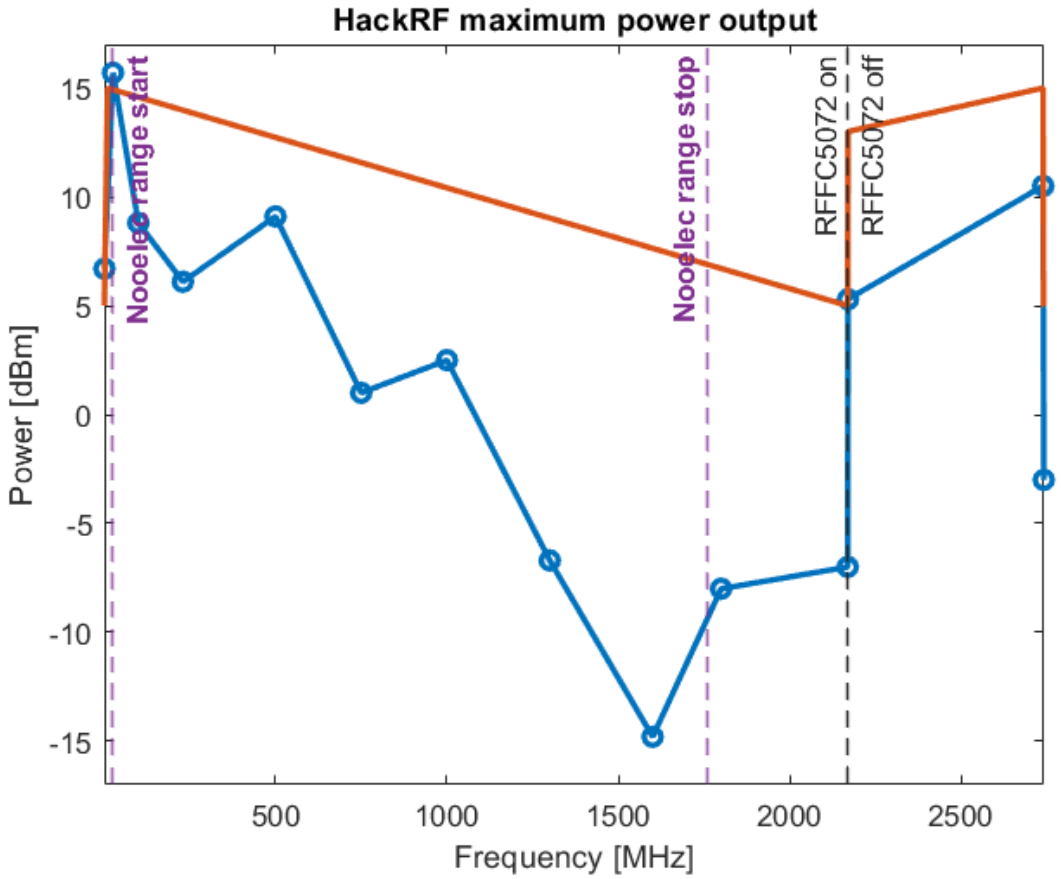


Figure 3.11: HackRF maximum power output.

In any case, disregarding the concerns about signal quality, the maximum power output of the HackRF, that is with the maximum IF gain and with the output power amplifier enabled, was measured and compared against the declared one[16]. The results, shown in figure 3.11, partially confirm the information reported in the HackRF documentation: although the maximum output power does not follow a linear trend with respect to frequency, both the general behavior and the specified limit values are largely met, within a tolerance of approximately -3dB. Nevertheless, a significant variation between the maximum and minimum output power is observed, amounting to about 30dB. The minimum occurs around 1600MHz, which corresponds to the same region where the Nooelec NESDR exhibits a marked drop in sensitivity. This combination may pose particular challenges when using the spectral region around 1600MHz for saturation and distortion measurements.

In addition, the user should pay particular attention when using the final stage amplifier: the Agilent/Broadcom MGA-81563 amplifier used is particularly sensitive to ESD (electrostatic discharges) and to power reflections due to impedance mismatches. The lack of an input protection circuit on the HackRF port, combined with this component flaw should be kept in mind by the user, to avoid damaging the amplifier. To avoid this risk, it is sufficient to turn off the signal output or power amplifier before any operation on the attached load and avoid touching the RF connector with bare hands.

4 | Analysis software

Having discussed the hardware utilized in this project and understood its capabilities, a software capable of orchestrating it and to exploit the provided data to perform useful analysis has to be developed in order to prove the validity of the system. This software has been called "CotSA companion app" and it is developed in MATLAB language leveraging the object oriented programming paradigm. While the software offers some useful measurement and tests, like two tones test, a power transfer curve tracer capable of measuring the P_{1dB} parameter, digital modulation analysis, a spectrum analyser window and a signal generator, its primary purpose is to demonstrate the feasibility and usefulness of such system. For this reason, the software is designed to foster expandability through the use of a clear and logical structure.

4.1. An object oriented programming paradigm

While for simpler programs a procedural programming (to give an example, C programs) approach might be sufficient and ideal for its low complexity, a beginner should not be scared of object oriented programming (Java code as an example), as after grasping few key concepts, it can also be easier to approach[39].
With basic procedural programming, the programmer usually writes the code as a series of procedures that are functions containing variables and sequential instructions to operate on them. Variables can exist as global variables, meaning they can be read and modified by all parts of the code as they exist globally and can be viewed by any part of the program; or local variables, that exists only within the scope of a single function. An example of high level procedural pseudocode is the algorithm 4.1.

Algorithm 4.1 A procedural programming example

```
1: GlobalVariable = 10;        %This variable and the next one can be seen by
2: AnotherGlobalVariable = 20; % every part of the program.
3: PROGRAM BEGIN
4:    First instruction
5:    Second instruction
6:    call Function1      % The "Function1" code will run after this line.
7:    Third instruction % After a function end, the program keeps executing the
8: END                    % next line after the function call.
9: Function1:
10:   LocalVariable       % This variable can not be seen by the main program.
11:   Function1 code
12:   Function1 more code
13: END Function1
```

With object oriented programming we assist at a logical shift: code is no more distributed in heaps as simple lists of instructions, but ordered in objects that are classes instances. But what are objects and classes? We can see an object as an ensemble of variables, called attributes, and functions, called methods, to operate on those variables. The variables are usually protected by external access (private variables) so that they can only be modified or retrieved calling methods. Classes are the structure of an object: exactly as in procedural languages a variable can be, let's say, an integer type, in object oriented languages, objects have class types.
While in procedural languages the code can be seen as a list of instructions, in object oriented programming, code represents objects interacting with each other through methods (algorithm 4.2).
Object oriented languages are powerful not only because the code is organised in a logical way and data are protected from external access, but also because this paradigm fosters code reusability. Objects and classes hide inside themselves the complexity of the operations are called to complete, exposing just simple methods to interact with the rest of the codebase. Thanks to these features, object oriented codes are easier to understand, to maintain and to expand with new features as we will see in this chapter.

Algorithm 4.2 An object oriented programming example

```
1: CLASS Dog
2:   FUNCTION __init__(self, name, age)      % What to do when a
3:     self.name ← name                      % new object is created.
4:     self.age ← age
5:   END FUNCTION
6:   FUNCTION bark(self)
7:     PRINT self.name + "says: Woof!"
8:   END FUNCTION
9:   FUNCTION show_age(self)
10:    PRINT self.name + "is" + self.age + "years old."
11:  END FUNCTION
12: END CLASS
13: PROGRAM BEGIN
14:   dog1 ← NEW Dog("Buddy", 3)      % Create an object of class "Dog".
15:   dog2 ← NEW Dog("Luna", 5)       % Create another object of class "Dog".
16: % Operate on the first object using its methods.
17:   dog1.bark()       % Output: Buddy says: Woof!
18:   dog1.show_age()   % Output: Buddy is 3 years old.
19: % Operate on the second object using its methods.
20:   dog2.bark()       % Output: Luna says: Woof!
21:   dog2.show_age()   % Output: Luna is 5 years old.
22: END
```

4.2. Language selection

Having decided to use an object oriented language, the next logical question is "which one to choose?". For their powerfulness, many exist and are used. To simplify the choice, the four most popular will be analysed: C++, Java, Python and MATLAB. These languages are not only the most popular, hence more supported by the community, but also usually taught or seen in various academic courses: as this project targets also students as users, selecting a language known also by them can be seen as a benefit.

C++ is a very powerful language that allows to write programs with excellent performances. Many of the used libraries, like hackrflib and libusb are natively written in C++. Unfortunately, because of the syntax and the way C++ code is structured, a C++ program can become very hard to understand. Moreover, C++ programs are usually

computer architecture dependent, meaning that different executables have to be compiled for each different operating system. Such operation can be quite hard because of the complex toolchain needed. For these reasons, the use of C++ code was quickly deemed unfeasible.
Java solves most of the C++ issues, relying on a well defined programming schema. The system compatibility issue is handled by the JVM (Java Virtual Machine), that can be described as a layer that interposes between the program code and the operating system, adapting the first to the latter. However, there are not well established library to interact with the chosen hardware.
The two main languages used in the scientific world are Python and MATLAB. From a functional point of view they are almost equivalent: both of them offer powerful library (called Toolbox in MATLAB and packages in Python) to facilitate the implementation of signal processing and arithmetic. On this regard, one might say that Python and MATLAB are competitors: they both offer a beginner friendly ecosystem with a complete documentation of the various code functions and an active community. From a coding prospective, both languages use dynamic typing, meaning that the user does not have to specify the variable type (e.g. integer, float, string), but it is the compiler to pick the proper type automatically. Both languages hide most of the software engineering specific task, leaving a beginner programmer the freedom to code without worrying of the coding technicalities. In this regard, an important difference can be found: while Python leaves to the programmer the choice on how to handle more technical and complex topics like multithreading, MATLAB completely hides the choice in its automatisms. Moreover, while both can be considered object oriented languages with scripting features, that means snippet of code can be run from a console during the program execution to alter its function in real-time, MATLAB does not implement some templates and functionalities typical of object oriented programming. This, as we will see in the next sections, will greatly impact the code structure.
Because of the many similarities, to help the language selection, it can be useful to also consider the development utilities, like the IDE (integrated development environment): the Python Foundation, which maintain the Python ecosystem, only provides a compiler. A code editor is provided by external companies and developers. While this enables a degree of flexibility for a programmer, a clear feature-set set cannot be established. On the other hand, MATLAB is a proprietary programming language controlled by MatWorks. The MATLAB editor is the only software available. This lack of choice could be seen, at a first glance, as a strong limiting factor. In practice, though, MatWorks focused on developing a complete editor, giving particular relevance at the mathematical part over the programming one, creating an environment where it becomes easy to debug and test

mathematical operations and processing.
Finally, it has to be noted that, being MATLAB a proprietary language, some licenses are needed. This could be a great impairment for many people not having access to an affordable way to get the required licenses and partially strides with the aim of this project. Despite this, MATLAB was the language chosen: the ease of developing and debugging signal processing code it has been considered more important at this stage to prove the feasibility of the idea. Coding considerations have been taken to facilitate a future porting of the program on Python and to obtain a truly free, libre open source software FSF compliant [27].
To conclude the programming language analysis, it is worth mentioning that Simulink, a MatWorks simulation environment, was taken into consideration, but quickly abandoned: while the graphical interface is helpful to allow a first glance understanding of the program, the implementation of the functional block did not allow a real-time operation of the system, rendering it useless.

4.3. Program structure

The program code is divided into several classes to maintain a good clarity of the code structure and facilitate maintenance and implementation of new features. The objects derived from the classes represent well defined components with clear and descriptive names as will be discussed in the next sections. Where possible, design patterns are used to better structure the code[6].
The main structure that shapes the code is the MVC pattern (model-view-controller) where program functionalities can be grouped in 3 macro categories: the model group represents the core of the program, where, as an example, signal processing is carried out; the view group, as the name might suggest, holds the code to interface with the program; the controller group contains the code that links the model to the view by preparing data to be displayed, performing user input sanitization and keeping coherence between the state of the model and what is displayed to the user. It has to be noted that this grouping is not strictly done dividing code in different classes, but a logical approach was followed. Infact, MATLAB provides widgets to easily build a graphical interface that already integrate some basic control functionalities. Where it was more logical, like in the receiver and transmitter classes, the model, controller and view parts are tightly coupled together, but still distinguishable when reviewing the code keeping in mind the definition of MVC pattern.
MATLAB does not implement the concept of classes interfaces and abstract classes, so these common features of object oriented programming can not be employed to enable

polymorphism, the ability of a class to represent different types of objects. A workaround to this lack of features was designed and will be discussed in section 4.8.
Visually, the program is divided in two tabs (figure 4.1): the "basic" tab holds the controls of the HackRF and Neesdr, a selection of test patterns and a spectrogram view of the received signal with some basic measurement features; the "measurement" tab displays specific automatic measurement and test functions, that are automatically selected depending by the signal generated within the "basic" tab.

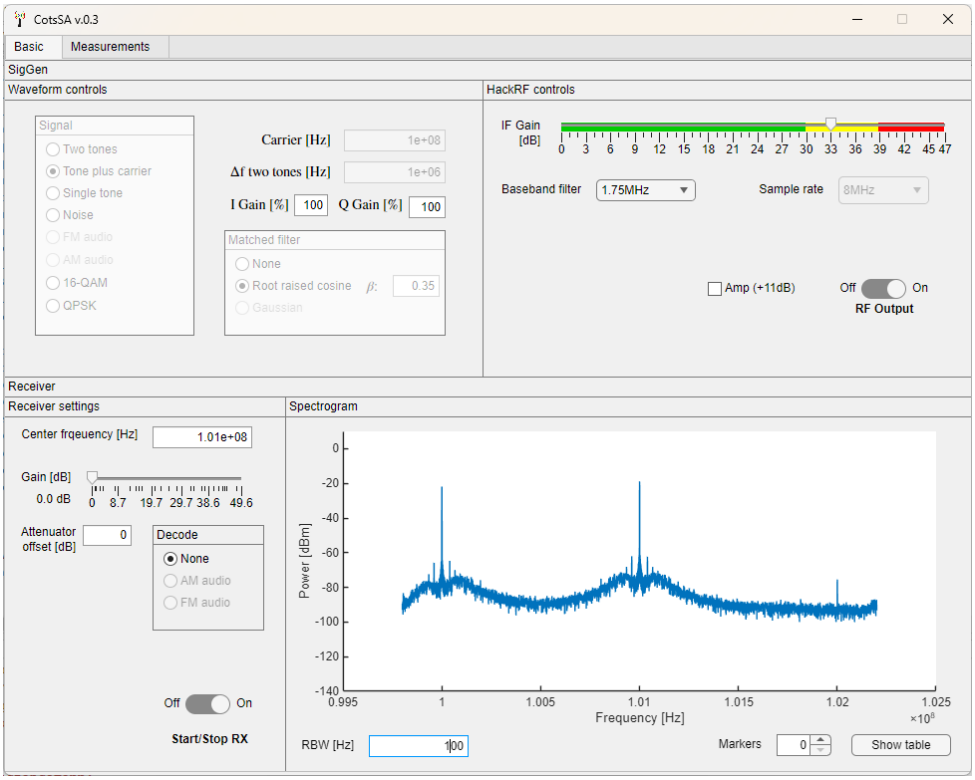


Figure 4.1: CotSA graphical user interface.

Each tab is composed by different objects: the "basic" tab content is handled by the transmitter_class and receiver_class objects, while the "measurement" tab is handled by a different object depending on the measurement performed. The entry point of the program is the "main" class, which extends the MATLAB matlab.apps.AppBase class, part of the MATLAB UI (user interface) framework. The main class is responsible for creating the main program window, handling hardware errors, like a missing SDR, and orchestrate the various objects.

4.4. Transmitter class

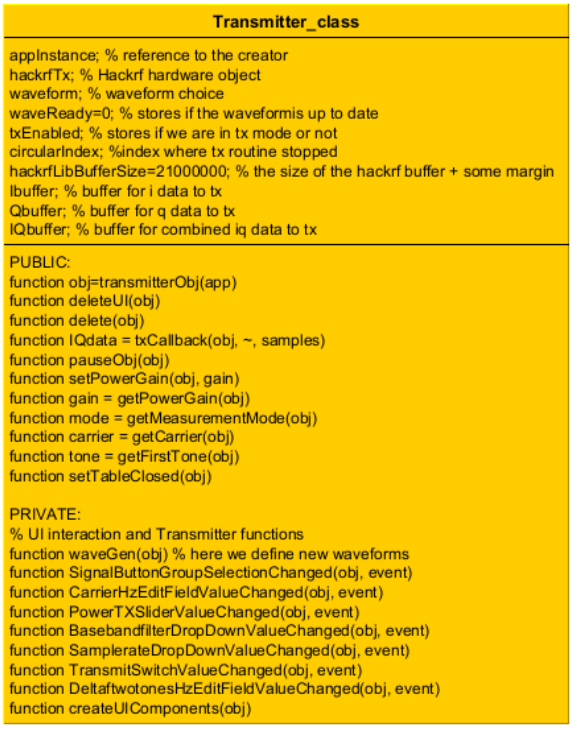


Figure 4.2: Transmitter class UML[29] diagram.

The transmitter class (figure 4.2) has the complex task of managing the HackRF hardware, generate the requested signal and sending them to the transmitting SDR, while keeping the user interface updated. Several methods are used to achieve these goals: following the MVC pattern logic explained in section 4.3, a distinction can be made. the createUIComponents method, called by the object creator, is responsible for creating the GUI (graphical user interface) elements regarding the transmitter object. At each element can be associated several control function that can either perform checks on the input values or update the model behaviour. Some of these functions are identified with a pattern name like "*GUIElement*Changed": these methods are called when a new input has been made and are responsible to check for the correctness of the input and to reflect it to the model by calling specific instruction in an ordered way. As an example, the "PowerTXSliderValueChanged" method is called every time the VGA gain slider is moved and is responsible to ask the HackRF to vary the VGA gain of the MAX2837. The most complex control function are the ones related to the signal selection and sampling rate, as affect many model's parameters. The control methods are classified as private: this means that methods can be called only by other function in the object itself, and

not by external objects. The model of the object is represented by the various attributes, among which the most important are the IQbuffer and the hackrfTx object; and methods, among which the waveGen function and the txCallback function.
When the transmitter_class object is created at the start of the program, a hackrf interface class (chapter 3.3) object is created. If this fails because no HackRF physical device can be found in the system, the program execution is halted. Otherwise, both the graphical interface and the SDR device are configured with standard default parameters. When the signal generation is activated by a user through the GUI, if a valid waveform is already present in memory (IQbuffer), that is indicated by the waveReady variable, a recurrent timer within the hackrfTx object is started. Each time the timer fires, the txCallback is called with parameter "samples" indicating the number of required samples to be generated and to be sent to the HackRF device. These samples are, therefore, prepared by the txCallback method and returned to the caller. The txCallback method offers different ways of producing the requested samples, whether they are constant signals, cyclic signals or generated real-time. As an example, the single tone is generated just by sending constants with the requested amplitude to the HackRF interface, which, when modulated, will create a tone at the center frequency. The noise signal is generated in real-time to guarantee low correlation among samples, that would not be possible using a cyclic buffer. The noise is generated by using a Gaussian random distribution, so that the extracted values and thus I/Q samples are insides the unitary circle, so that they can be scaled to utilize the full DAC range. The other signals are generated before the signal transmission is activated by the waveGen function and stored in a cyclical buffer called IQbuffer, so that the signal is repeated every few seconds. This allows to generate and use complex signals and modulation schemes while keeping the transmission process responsive: a real-time generation of such signals would require too much resources to be successfully made. A noticeable effect of this design choice is that, when a new waveform requiring to be generated before transmission is requested, it can elapse some time before the RF output is actually activated: during this period, the program, thanks to the waveGen function, is filling the IQbuffer.
Additionally, the txCallback function allows to artificially add to every signal type I/Q impairments through the graphical interface.
The waveGen function is called each time the signal transmission is activated but the waveReady flag is set to 0, as this means that the waveform already present in the IQbuffer is not up to date with the latest user inputs. Elements that trigger a waveform regeneration are a change in the requested waveform, a change in the sampling rate and changing specific to the requested signal (such as the spacing in the two tones signal).
The two tones test signal is generated by creating a sinusoid at the frequency $\frac{DeltaTwoTones}{2}$,

as after hardware mixing, the resulting image frequency would be at a frequency of $f_c - \frac{DeltaTwoTones}{2}$, while the generated sinusoid at a frequency of $f_c + \frac{DeltaTwoTones}{2}$, effectively creating a two tones signal with tones spaced $DeltaTwoTones$, as defined by the user. This sinusoid is used as In-phase component. To have a greater amplitude signal, that becomes the complex envelope amplitude instead of the single in-phase component amplitude, the Quadrature component can be easily added by using the same in-phase baseband signal used. While this method would theoretically lead to a sinusoid signal with a phase shift dependent by the two components amplitude (45° if equal amplitude), the lack of a shared time reference among the devices employed solves this potential issue. In chapter 3.6 was proposed a method to generate two shifted tones to cover the central spike given by the oscillator leakage and the possible DC offset. A constant signal with half normalized range (0.5) is used and summed to a sinusoid with maximum amplitude of ± 0.5. In this case the frequency of the sinusoid is exactly $DeltaTwoTones$, as the image frequency will not be exploited. If the aforementioned signal is used as in-phase component, without considering the quadrature one, the results after mixing are three tones: a tone at the carrier frequency, generated by the constant component in baseband, a tone at $f_c + DeltaTwoTones$ generated by the sinusoid component and another tone at $f_c - DeltaTwoTones$, that is the image frequency generated by the mixing process. This last tone is undesired in this case, but can be effectively removed by further digital signal processing. This is obtained by applying the Hilbert transform[2] to the aforementioned signal and using the analytic signal obtained as new in-phase and quadrature component. In practice, considering I_r as the original signal, the complex analytical signal A_c is obtained as:

$$A_s = I_r(t) + j\mathcal{H}(I_r(t)) \tag{4.1}$$

where $\mathcal{H}$ is the Hilbert transform. The Hilbert transform advances an arbitrary signal negative frequencies phases by 90° while shift positive frequencies phases by -90°, the transformed signal is then shifted of $+j$ and added to the original signal to obtain the complex analytical signal. The effect of this operation is that the spectral content of the analytical signal lies in the positive Nyquist interval, causing the removal of the image frequency in passband, after mixing. The use of the Hilbert transform, in practice, allows the generation of a SSB (single side band) modulated signal, where the power is focused on the right side, so towards positive frequencies, respect to the center frequency. To effectively use the analytical signal generate, its real part is used as quadrature component, while the imaginary part is used as in-phase component.
All the tones signals are passed through a digital FIR (finite impulse response) filter, before being saved to the IQbuffer vector.

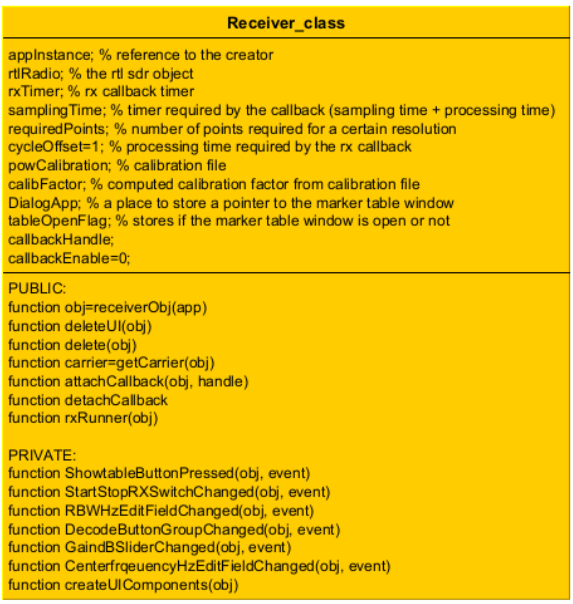


Figure 4.3: Receiver class UML[29] diagram.

Also the digital modulation signal are generated by the waveGen function: the message signal is a random binary sequence based on the PRBS-7 (pseudo-random binary sequence, order 7)[37]. The available modulations are 16-QAM (4 bits/16 symbols quadrature amplitude modulation) or QPSK (quadrature phase shift keying) to test both non-constant and constant envelope digital modulation schemes. The signal is generated by leveraging on the respective QAM[33] and QPSK[34] MATLAB modulator modules. The constellation symbols are arranged with a Grey-coded ordering, with a symbol rate fixed to 100KHz. The modulated signal can be then used as it is, without additional filtering, or, more correctly, be filtered through a root raised cosine filter, with roll-off factor selectable by the user. The complex signal obtained is placed into the IQbuffer to be used by the txCallback routine.

4.5. Receiver class

The receiver class (figure 4.3) is the class devoted to handle the Nooelec Nesdr device, acquiring the signal, displaying the results and passing the signal to the measurement classes for further analysis. Also in this case, as it was for the transmitter class (section 4.4), a distinction can be made among methods handling the view layer, control layer and the model. Moreover this class uses an additional helper object, as it will soon be described to display some basic measurement about the acquired signal.
Also in this case, the createUIComponents function is called at the beginning of the

program to generate user interface elements regarding the receiver object. The available control are to set the center receiving frequency and the gain. It has to be noted that, while the gain of the transmitter can be selected in steps of 1dB, the receiver has predefined steps with non constant spacing[1]: the GaindBSliderChanged callback function rounds the requested gain to the nearest available value and updates both the slider in the GUI to reflect the actual used value and the hardware SDR gain. The tuning process is handled by the CenterfrequencyHzEditFieldChanged callback: when the user finishes to enter a new center frequency, the function is called and the new frequency is sent to the R820T tuner. After this command, the Nesdr receives the samples from the new frequency.
The resolution bandwidth can be modified by using the dedicated input field. After a new resolution is defined, the RBWHzEditFieldChanged function is called: this function has the important task of computing the required signal length to be acquired in order to have the desired resolution after performing the fast Fourier transform and crossing into the frequency domain. Considering the sampling rate S_r of 2.4Ms/s, the required samples necessary to have a certain FFT (fast Fourier transform) resolution, equal to RBW, can be found so that:

$$FFTsamples = \frac{2.4Ms/s}{RBW} \tag{4.2}$$

taking care to round the result to the nearest integer, as it would make no sense to talk about partial points. Considering that the sampling rate is fixed, it is possible to know how much time is necessary to capture a single samples. Knowing from 4.2 the points necessary, it is trivial to find the acquisition length:

$$Length[s] = \frac{1}{2.4Ms/s} \cdot FFTsamples \tag{4.3}$$

When the acquisition process is started, by using the appropriated button on the user interface, a cyclic timer is activated. The timer calls the rxRunner function each time it fires. The rxRunner function is the core function that handles the acquisition process: each time is called, a length of signal equivalent to what found in equation 4.3 is acquired at 2.4Msamples/s and stored in a vector. It has to be noted that the digitalized signal is already at baseband, and is provided in its normalized in-phase and quadrature components.
Every real world acquired signal is affected by a DC (direct current) offset, and the one from the Nooelec makes no exception: this DC component can be quite impactful on the measurements as, after the FFT, it presents itself as a undesired great magnitude spike

[1]Nooelec Nesdr available gains: 0dB, 0.9dB, 1.4dB, 2.7dB, 3.7dB, 7.7dB, 8.7dB, 12.5dB, 14.4dB, 15.7dB, 16.6dB, 19.7dB, 20.7dB, 22.9dB, 25.4dB, 28dB, 29.7dB, 32.8dB, 33.8dB, 36.4dB, 37.2dB, 38.6dB, 40.2dB, 42.1dB, 43.4dB, 43.9dB, 44.5dB, 48dB, 49.6dB.

that can skew the graphs scales, so usually it is desirable to remove it. This can be done either in the frequency or time domain. In the first case, an high pass filter with a very low cut-off frequency can be used, effectively acting as a DC block. Using such filter requires some attentions to its design: the passband portion of the filter should be as flat as possible an should not introduce noticeable phase delays to not alter the signal of interest. A much easier approach can be adopted in the time domain, especially in this case, where the time domain signal is windowed: by simply computing the mean value of the acquired signal and removing it, the problem is solved effectively. It has to be noted that, either way, the resulting signal is affected, meaning that not only the unwanted DC offset, but also the part of the signal that lies at 0Hz frequency in the baseband signal (center frequency in passband) is removed. Usually this does not impact significantly the results of a measurement or transmission.
After removing the DC offset, the signal is ready to be viewed in the frequency domain. This is achieved applying the FFT. The DFT (discrete Fourier transform) algorithm is optimized to work with a number of samples equal to a power of two: when enough samples are not available, the signal can be padded with zeros to reach the closest power of two. Aiming to preserve the actual requested resolution without rounding, this was not done, at the expense of computational efficiency. The result obtained is the power spectral content of the signal. The scale is converted from linear to logarithmic. To obtain a calibrated power measurement, a lookup table is used: in chapter 2.9 a program to obtain a calibration table was described. This table contains for a series of frequencies equally spaced a calibration factor, that is a scaling factor that indicates the relation between the SDR readings and the actual power used. By interpolation, each frequency between the tabulated points can be found. To obtain a calibrated power reading it is sufficient to perform a simple sum:

$$P_{dBm} = P_{dB} + CalFactor - RTLGain \tag{4.4}$$

The obtained calibrated result is then displayed into a spectrogram. The function is also responsible to pass the acquired data to test objects in the measurements tab. This is accomplished by passing the test object function handler to the receiver object when the first is created. Each time the rxRunner function is executed, the handler passes the data to the test object to perform additional analysis. As said in the previous section, MATLAB hides the threading environment: this means that while test functions would highly benefit from running independently of the receiver function, due to the different elaboration timing, the lack of easy thread access forces the test functions to be executed sequentially in the same rxRunner process and must be kept as light and fast as possible

to not impact the reception process altering its timings. Regarding the spectrogram, the Receiver panel offer an additional feature: the possibility to add markers to the measured signal: this is done by finding the local maxima present in the spectrum and, after ordering them from the biggest to the smallest peak, keeping only the first requested ones. A separate windows containing a table can be opened to show frequencies and amplitudes of each markers. From a programming standpoint, such windows is created relying on a singleton pattern: this pattern allows only one object of a defined type to be created. In this specific case, just a markersTable object can be created and, if already existent, is returned instead of generating a new one. In this way duplicates of the markers table window cannot be spawn.

4.6. Measurement tab and test framework

As threads in MATLAB are not implemented in the same way of most programming languages, but in pool of workers that may or may not share a single processor thread, predicting their behaviour can be even more difficult that what already is in other languages. For this reason, the test/measurements framework is designed to run in the same thread of the rest of the program. This is clearly an important limitation, as the computation performed for the test must run in quasi-realtime, between each acquisition cycle, that lasts around one second. To alleviate this limitation, the transmitter and receiver classes offer some methods to adapt their configuration to better suit the test requirements: as an example, the transmitter class offer methods to share its configuration and to externalize the output power control, while the receiver class offer methods to let other classes control its tuned frequency and the acquisition timings.
The test type selection is based on which signal is requested to the signal generator/transmitter class: when the user passes from the basic tab to the measurement tab (figure 4.1, top-left) by clicking the apposite label, the main object, which is the root object of the program, checks with an apposite method which signal is selected on the transmitter object; knowing that, it can create the more appropriate measurement object that dynamically generates the graphical interface for the selected test in the measurement tab.
The test is enabled by the rxRunner method of the receiver class: the specific test class must provide a function handler that, when called by the rxRunner, receives as an argument either the raw acquired samples or the calibrated power spectrum to operate on. It has to be noted that any computation on the passed data happens in the called function itself, temporarily blocking the rxRunner execution.
Considering the reasons just exposed, a schema for a generic measurement class can be traced. Each class must contain at least:

- a GUI generation method to populate the user interface when created;
- a GUI deletion method to release the user interface when another measurement is selected;
- a callback method to be invoked by rxRunner to perform the measurements.

4.7. Two tones test class

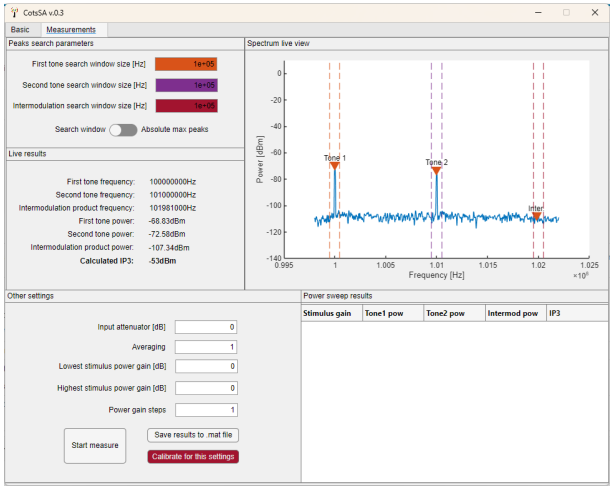


Figure 4.4: Two tones test user interface.

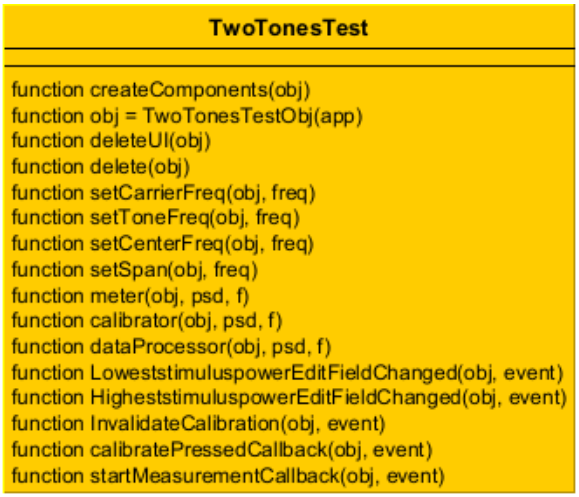


Figure 4.5: TwoTonesTest class UML[29].

The two tones test is available in the measurement tab when the two tones or tone plus carrier signals are selected in the SigGen panel (transmitter class, section 4.4). The user interface is divided in two main parts: the first consist in a live view to quickly perform a first measurement, the second comprehend settings to perform a power sweep and track the intermodulation tone changes to perform a calibrated measurement. This distinction can be found also in the code, where, while there are some strong similarities between the functions of the two parts, the logic behind changes. In the live view mode, user selects the desired tone power by using the gain in the "basic" tab. After attaching the DUT to the measurement system (figure 1.1a) and enabling signal generation and acquisition the tones should be visible in the spectrum live view of the "measurement" tab. Within the "peaks search parameters" panel, user can select between two peaks search methods: the "absolute max peaks" search for three peaks in the whole spectrum, assuming that the two main ones are the two generated tones and the third is the intermodulation product; the "search window" method search peaks only inside windows of defined bandwidth. These windows are centered around the generated tones frequencies and the theoretical

intermodulation tone frequency, computed as:

$$f_{int} = 2 \cdot f_2 - f_1 \tag{4.5}$$

where f_2 is the highest frequency tone, f_1 is the lowest frequency tone and f_{int} is the intermodulation product frequency.
After measuring the power level of each of the three peaks, the live results are obtained following equation 2.1. The measurements are updated each time a new sample block is acquired by the receiver invoking the dataProcessor function.
The calibrated measurement mode uses a similar method, but some workarounds were necessary to avoid the use of threads in an explicit way. Opposite to the live view mode, after configuring the parameters, the user attaches the HackRF output directly to the Nooelec input and presses the calibrate button: during this routine, all the requested power level are swept and results are saved. This allows to have not only the two tones power readings, but also the HackRF intermodulation product power value (see section 3.6). After having performed this step, the DUT (device under testing) can be attached as in the live view method (figure 5.2 as reference). When the start measure button is pressed, the same power sweep is repeated. To calibrate the results, by keeping in mind that by hypothesis the measurements are carried out far enough from the P_{1dB} point, the amplifier gain is computed by comparing how much the two used tones have been amplified. Knowing this and the HackRF intermodulation product, it is possible to calculate its power growth and compensate for it in the final results:

$$\begin{cases} gain = P_{f_1} - P_{f_1 cal} \\ IP3 = \dfrac{P_{f_1} + 2 \cdot P_{f_2} - (P_{int} - (gain \cdot P_{int_{cal}}))}{2} \end{cases} \tag{4.6}$$

The results are then displayed in the dedicated "power sweep result" table.

4.8. Digital modulation analysis class

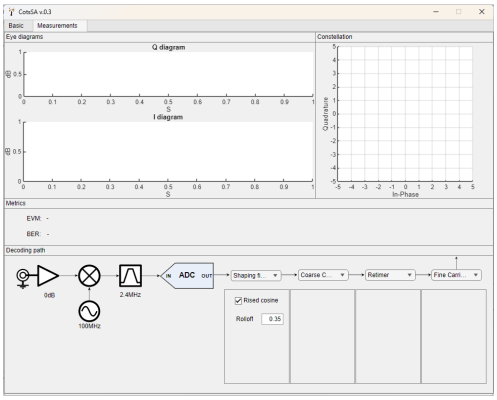


Figure 4.6: Digital modulation test user interface.

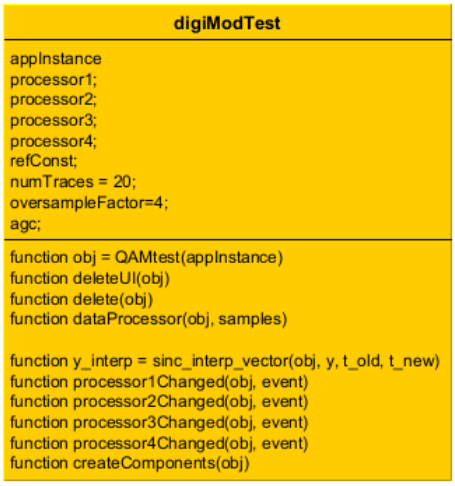


Figure 4.7: Digital modulation test class UML[29] diagram.

The digital modulation analysis class acts as a demodulator for the QAM and QPSK signals generated by the transmitter class. It is designed to provide a graphical view of the processing chain. To obtain a flexible and reconfigurable signal processing chain, an additional "filterClass" was designed to abstract the actual object used to process the signal and provide a common software interface. Each of the four position in the processing chain can be arbitrarily used with one of the following functions:

- Pass-through
- DC block filter
- Low-pass filter
- Matched filter
- Carrier frequency offset compensator
- Retimer
- Phase offset compensator

The processing flow is shown in figure 4.8: after the processing blocks, that can be reconfigured by the user through a drop-down menu, the symbols are plotted in two eye diagrams, one for the in-phase component and the other for the quadrature component.

The raw, not decoded symbols are compared against the reference constellation to calculate the average error vector magnitude (EVM).

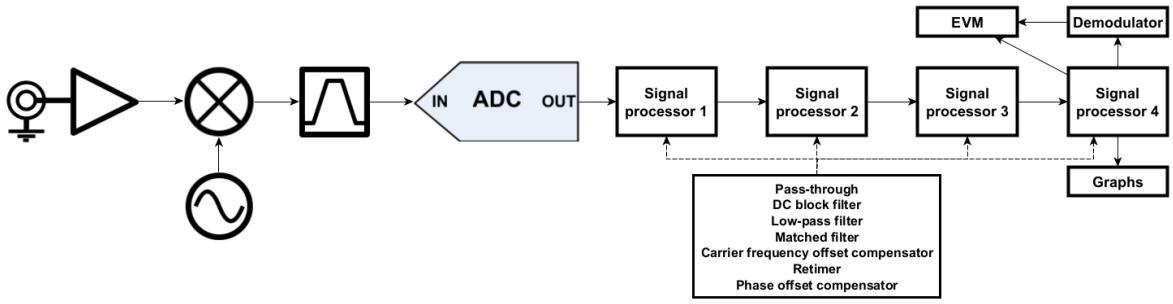


Figure 4.8: Signal processing flow of the digital modulation analysis.

4.9. Single tone analysis class - Power transfer curve tracer

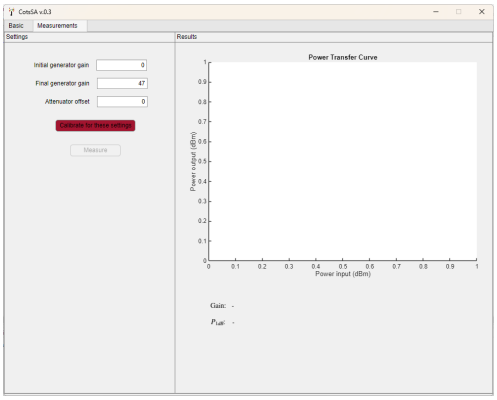


Figure 4.9: Single tone analysis user interface.

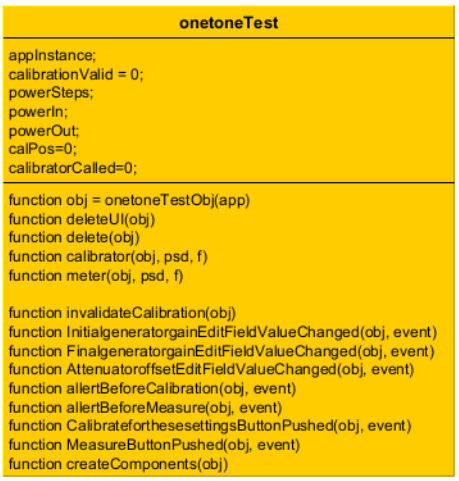


Figure 4.10: Single tone analysis class UML[29] diagram.

The single tone analysis class provides a way to trace the input/output power transfer curve of an amplifier (the DUT) and measure its 1dB compression point (P_{1dB}).
The interface of the test is very simple, as the automations are responsible for most of the work. The power sweep is done in two phases: the first one is a power calibration sweep without the DUT, while the second is the actual sweep with the DUT attacched. As discussed in section 3.7, no HackRF output power measurement is directly available,

and the power control is done through the HackRF IF gain setting. This means that, before being able to perform any real measurement, for the selected tone frequency, a relation between a gain value and the tone power content associated with that gain must be established. This is obtained connecting the HackRF output directly to the Nooelec Nesdr input: by sweeping the IF gain range and recording the output power measured at each setting, it is possible to recreate a reference table that ties a gain setting with an output power value. This operation is done through the red calibration button, that turns green when a valid calibration is present in memory. When a valid calibration is present in memory, the "measure" button is enabled: after attaching the DUT between the HackRF and the Nesdr, the button can be pressed to start the measurement. During this sweep, the DUT output power is recorded. Through the calibration table, a relation between the DUT input and output power can be established. The results are plotted in the power transfer curve graph. The utility also provides three measurement: an estimation of the DUT gain, the estimation of its P_{1dB} and of its IP_{1dB}. The gain is estimated using the first three points of the curve: considering P_{nOUT} the n-th measured power at the DUT output port and P_{nIN} the corresponding input power,

$$\begin{cases} Gain_1 = P_{1OUT} - P_{1IN} \\ Gain_2 = P_{2OUT} - P_{2IN} \\ Gain_3 = P_{3OUT} - P_{3IN} \\ Gain = \dfrac{Gain_1 + Gain_2 + Gain_3}{3} \end{cases} \tag{4.7}$$

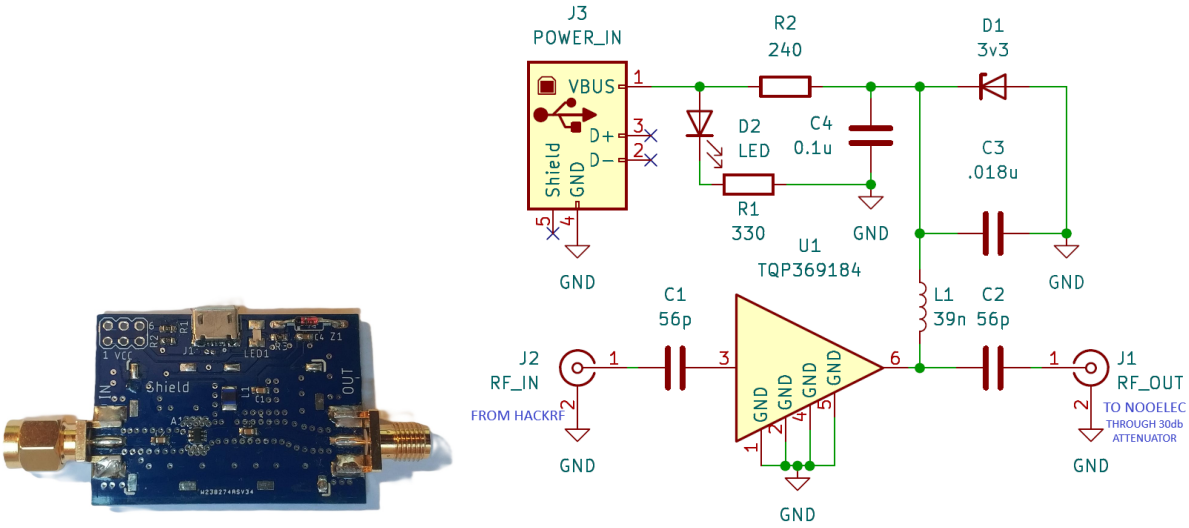
provided that the used point are far enough from the 1dB compression point.
The P_{1dB}, that is the point at which an amplifier's output power, when driven by an increasing input signal, deviates by 1 dB from its ideal linear gain, is found by searching the point that satisfies the condition:

$$P_{out} < P_{in} + gain - 1 \tag{4.8}$$

The IP_{1dB} that is the stimulus power at the P_1dB point is also provided.

5 | Results analysis, conclusions and further developments

To test the validity of the measures obtained with the designed system, it was chosen to test a commercial component and to compare the obtained results against the datasheet declared figures.
The component used as device under testing (DUT) was the Qorvo TQP369184 20.3dB RF amplifier. This component was picked for its great availability at low cost, but more importantly, because of its detailed datasheet. The TQP369184 is already input and output matched, so only few components to realize the power section, such as DC-blocking capacitors, a bias resistor, and an inductive RF choke are required for operation, making easy the realization of an evaluation board. The schematic of the test circuit is provided in figure 5.1b.



(a) Amplifier board.

(b) Qorvo TQP369184 test circuit.

Figure 5.1: Qorvo TQP369184.

5.1. Two tones test

Looking at the Qorvo TQP369184 datasheet, it is said that the ouput IP3 is typically 28.5dBm, with a minimum value of 25.5dBm[19]. The Qorvo's test conditions are two tones with a power of 0dBm for each tone, spaced 1MHz.This conditions are within the CotSA system capabilities. The device was connected as indicated in figure 5.2, using a 30dB attenuator on the amplifier output to avoid saturate the Nooelec ADC.

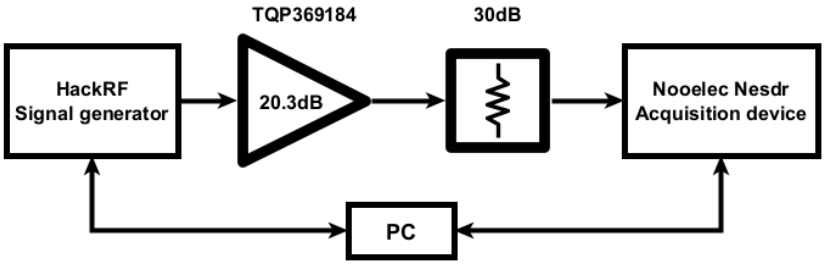


Figure 5.2: DUT connection diagram.

To obtain a 0dBm/tone output from the Qorvo amplifier, the HackRF gain was set to 33dB, equivalent to an amplifier input power of -20dBm/tone: this value also confirm the Qorvo TQP369184 declared gain of 21dB.

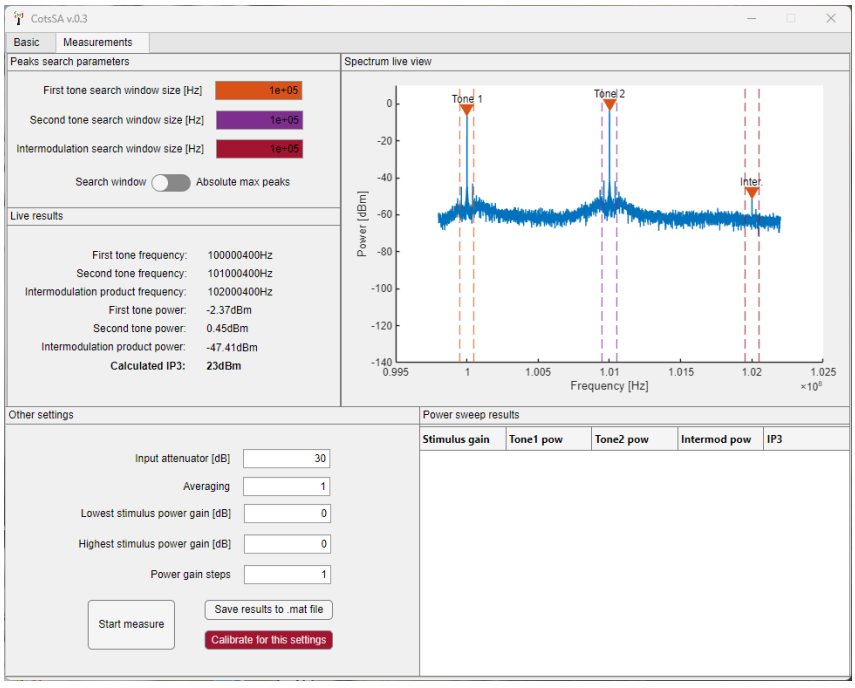


Figure 5.3: IP3 test results.

The gain setting, as suggested in section 3.7, was found by connecting the HackRF output directly to the Nesdr input and tweaking the HackRF gain value through the dedicated slider until the desired value was found on the spectrogram using the markers function(see section 4.5).
For this test, the "carrier plus tone" signal was used (the generation process is described in section 4.4, whereas the underlying rationale is discussed in section 3.9), but a similar result can be obtained with the "two tones" signal.
The results of the test, visible in figure 5.3, show an IP3 value of 23dBm, only 2.5dB from the declared figure. This discrepancy is probably due to the non perfectly equal tones power content: while the second tone has a power of almost exactly 0dBm, the first one as 2dB less than the required.

5.2. P_{1dB} measurement

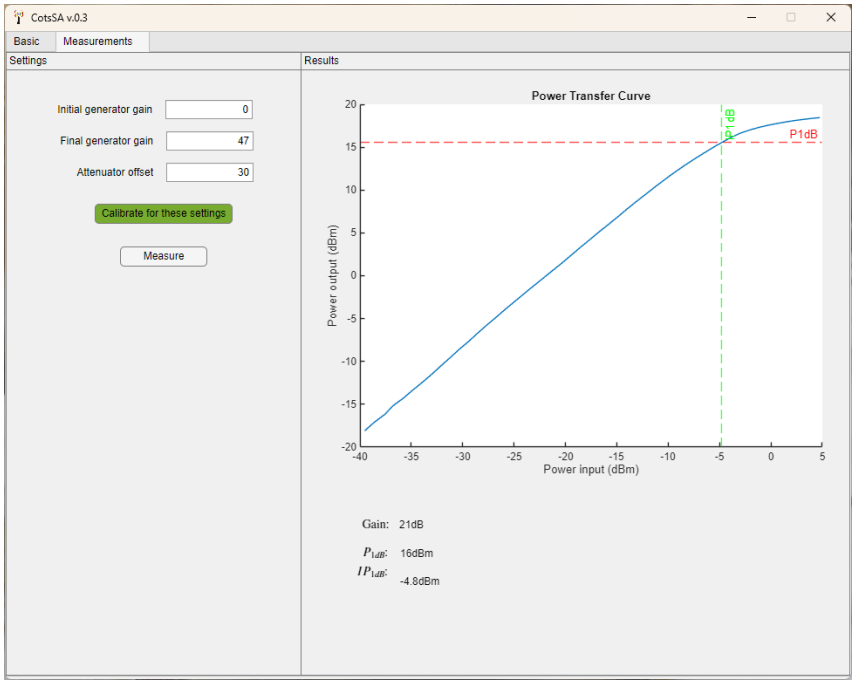


Figure 5.4: P_{1dB} test results.

Connecting the Qorvo TQP369184 as in figure 5.2, a power sweep was conducted, as described in section 4.9. A 30dB attenuator before the Nooelec Nesdr was used to prevent ADC saturation. The results, shown in figure 5.4 (raw data available in appendix A) confirm what indicated in the datasheet: at the frequency used, 1GHz, the datasheet shows a gain of 21.8dB, while the test measured 21dB. The datasheet declared P_{1dB} is 15.5dBm,

the experimentally obtained value is 16dBm, showing a difference of only 0.5dB.
This results confirm the capabilities of the designed system to perform accurate measurements that are confirmed by the manufacturer provided data.

5.3. Conclusions

This thesis project has explored the capabilities of two low-cost commercial SDR devices which, when paired with custom-developed software, form a practical and affordable measurement system, friendly named CotSA, (Commercial-off-the-shelf analyser, with reference to the two general purposes commercial SDR used).
Through a careful study and characterization of the hardware and a purposely developed MATLAB software, this project demonstrates that a meaningful signal generation and analysis can be achieved without resorting to complex and expensive lab-grade instruments, enabling students, enthusiasts, and interested individuals to gain hands-on experience with RF systems.
The motivation that sparked the idea behind this work, indeed, stemmed among others from the author's personal experience of the difficulty in hands-on experimentation with concepts studied in theoretical university courses, a difficulty that is objectively confirmed by the limited access to measurement instruments for hobbyists, students, and educators, mainly due to economic reasons and limitations: high-quality spectrum analysers and signal generators are often cost-prohibitive and delicate, limiting hands-on experimentation and independent learning. CotSA addresses this challenge by offering a modular, extensible system with a bill of materials of approximately €150, less than a tenth of the cost of many entry-level commercial solutions.
As acquisition device, the Nooelec Nesdr Smart V5, based on the RTL-SDR project, was chosen. The receiver was thoroughly tested in terms of frequency response, gain linearity, stability, input impedance (S11), and intermodulation distortion (IIP3). Despite its inherent limitations, such as modest ADC resolution and narrow dynamic range, strategies were implemented to overcome them: calibration routines using a reference signal generator enabled accurate compensation for frequency-dependent gain, and a calibrated spectrogram acquisition software was devised. Temperature stability tests and impedance mismatch analysis using VNA measurements provided further insight into the device's practical performance limits. During these stability tests, conducted varying the temperature of the SDR device by using ice, some instabilities in the form of loss of frequency tuning were highlighted at room temperature. Using a VNA, it was discovered an unexpected strong relation between input impedance and the SDR input gain setting.
On the transmission side, the HackRF One was employed to generate test signals. The

HackRF is a half-duplex software defined radio device. Its flexibility, open-source nature, and wide tuning range (1 MHz to 6 GHz) make it an ideal candidate for low-cost RF signal generation. The system relies on complex baseband signal generation in software, which is then upconverted to the desired RF frequency via the HackRF's internal analog mixing stages. Although this architecture has certain limitations, such as the presence of spurious tones and the lack of a direct passband DAC, it has proven to be sufficient for a wide variety of educational and experimental scenarios.

A major contribution of this thesis was the development of a custom MATLAB interface that replaces the old and abandoned community-developed libraries. This new interface provides a more stable and flexible environment, supporting dynamic waveform generation through callback mechanisms and enabling real-time control over transmission parameters such as frequency, gain and filtering. These improvements make the HackRF a much more user-friendly and adaptable signal generator and generic SDR device not only within the CotSA framework, but also towards other MATLAB developers, as the library has been publicly released.

As the Nooelec Nesdr Smart V5, also the HackRF underwent a series of tests to be properly characterized. During the testing phase, it was possible to fully assess the limited spectral purity, when compared with state-of-the-art instrumentation, and the presence of spurious emissions due to internal mixing artifacts, DAC quantization noise, and imperfect filtering. During testing, a notable variation was observed in the performance of different HackRF units: one board, for instance, exhibited significantly higher intermodulation distortion and local oscillator leakage, which degraded the quality of the transmitted signal. These discrepancies likely stem from the use of substandard or out-of-spec components, which are sometimes used in unofficial builds found on eastern markets to reduce costs. This underlines a key challenge when working with open hardware: while the design might be sound, the actual performance is only as good as the components used and the care taken in manufacturing.

Despite these limitations, the HackRF proved sufficient for generating a wide variety of test signals, including multi-tone waveforms and digitally modulated signals such as QAM and QPSK, with acceptable fidelity. A fundamental link between the two analysed devices is given by the developed software, written in MATLAB language. The software architecture is object-oriented and modular, fostering code maintenance and future features extension. To demonstrate the potential of the system, several test procedures were developed. These include two-tone intermodulation analysis, digital modulation analysis, such as 16-QAM and QPSK signal quality evaluation, and 1dB compression point automated measurement. Such functionalities make CotSA an ideal platform not only for learning RF fundamentals but also for performing basic performance characterization of

RF circuits and systems.
The system proved its usefulness analysing a commercial amplifier module, highlighting that, while not reaching the performances of a lab-grade instrument, it still can provide key informations to the user about the tested component.
Beyond the technical achievements, CotSA exemplifies how open hardware, combined with thoughtful software engineering, can democratize access to sophisticated measurement techniques, empowering users to experiment, measure, and learn autonomously, even in constrained settings.
It is also worth noting that the project strikes a balance between usability and performance. By making design choices that favour robustness, simplicity, and modularity, CotSA becomes suitable for use in home labs, workshops, and maker spaces, where ease of use and replaceability are crucial.

5.4. Limitations and future developments

While this thesis work has achieved many of its initial goals, several limitations and areas for improvement have emerged during development:

- Thermal stability of the RTL-SDR remains a concern at higher frequencies. Improvements in enclosure design and passive or active cooling could enhance performance in this range.
- Spectral leakage and spurious emissions from the HackRF, although acceptable for many applications, may limit its use in precision testing. Incorporating better filtering or pre-distortion techniques could be considered.
- Automation and GUI integration could be further developed. While the current interface is functional, some operations, such as calibration, are handled in separate programs and can be further integrated.
- Support for additional SDR platforms, such as PlutoSDR, another popular open source SDR platform by Analog Devices, could be added with minimal effort thanks to the modular software structure. This would allow users to trade off between cost and performance as needed.
- Additional testing and measurement routines can be implemented. As an example, it has been demonstrated[3] that a noise figure analyser is achievable with a similar setup as the one used in this project.

In conclusion, CotSA proves that an effective and educational RF measurement platform

can be built using low-cost commercial SDRs. While not a replacement for professional instruments in critical applications, it offers substantial value as a learning, prototyping, and exploratory tool. This work encourages the continued adoption of open hardware in engineering education and contributes to the broader effort of making RF experimentation more accessible.

Bibliography

[1] Agilent Technologies. *MGA-81563 0.1– 6 GHz 3 V, 14 dBm Amplifier Datasheet*. Broadcom Inc., Irvine, California, USA, 11 2005.

[2] K. Choi and H. Liu. *Hilbert Transform, Analytic Signal, and SSB Modulation*, pages 109–122. Wiley, 2016. doi: 10.1002/9781119060239.ch11.

[3] DL2ALF, DL8AAU, and DF9IC. Canfi, (c)heap (a)utomatic (n)oise (f)igure (i)ndicator with dvb-t stick, 12 2015. URL `https://github.com/dl2alf/CANFI`.

[4] D. J. et al. Siglent ssa3000x and ssa3000x-plus spectrum analyzers, 2015. URL `https://www.eevblog.com/forum/testgear/siglent-ssa3000x-spectrum-analyzers/`.

[5] O. et al. Rtl-sdr codebase. URL `https://github.com/osmocom/rtl-sdr`.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994. ISBN 0-201-63361-2.

[7] E. Kaashoek. tinysa ultra technical description, 10 2023. URL `https://tinysa.org/wiki/pmwiki.php?n=TinySA4.TechnicalDescription`.

[8] P. Kenington. *High linearity RF amplifier design*. Artech House, 2000. ISBN 1-58053-143-1.

[9] Keysight Technologies. *Keysight 2-Port and 4-Port PNA Network Analyzer N5227B 900 Hz to 67 GHz*. Keysight Technologies, Santa Rosa, California, USA, 10 2023. URL `https://cdn.teledynelecroy.com/files/pdf/wavemaster8000hd-datasheet.pdf`.

[10] K. Kundert. Accurate and rapid measurement of ip2 and ip3. *The Designer's Guide Community*, 13, 2002.

[11] Maxim Integrated. *MAX5864 Ultra-Low-Power, High-Dynamic and Performance, 22Msps Analog Front End Datasheet*. Analog Devices, Sunnyvale, California, USA,

10 2003. URL https://www.analog.com/media/en/technical-documentation/data-sheets/max5864.pdf.

[12] Maxim Integrated. *MAX2837 2.3GHz to 2.7GHz Wireless Broadband RF Transceiver Datasheet*. Analog Devices, Sunnyvale, California, USA, 2015. URL https://www.analog.com/media/en/technical-documentation/data-sheets/max2837.pdf.

[13] Maxim Integrated. *MAX2839 2.3GHz to 2.7GHz MIMO Wireless Broadband RF Transceiver Datasheet*. Analog Devices, Sunnyvale, California, USA, 2015. URL https://www.analog.com/media/en/technical-documentation/data-sheets/max2839.pdf.

[14] National Instruments. Ni-visa 2024 q3, 6 2024. URL https://www.ni.com/en/support/downloads/drivers/download.ni-visa.html.

[15] Nooelec. *NESDR SMArt RTL-SDR v5 Datasheet - Revision 1*. Nooelec, 4-2045 Niagara Falls Blvd, Wheatfield, NY, USA, 1 edition, 10 2022. URL https://www.nooelec.com/store/downloads/dl/file/id/111/product/249/nesdr_smart_rtl_sdr_v5_datasheet_revision_1.pdf.

[16] M. Ossmann. The hackrf one project. URL https://hackrf.readthedocs.io/en/latest/index.html.

[17] J. M. Pieper. Scpi and visa a valuable combination. *ACEA, Wierden, The Netherlands*.

[18] Qorvo Inc. *RFFC5071/5072 WIDEBAND SYNTHESIZER/VCO WITH INTEGRATED 6GHz MIXER*. Qorvo Inc., Greensboro, Carolina del Nord, USA, 10 2014. URL https://www.qorvo.com/products/d/da000735.

[19] Qorvo Inc. *TQP369184 DC – 6 GHz Gain Block Datasheet*. Qorvo Inc., Greensboro, Carolina del Nord, USA, 06 2020. URL https://www.qorvo.com/products/d/da000735.

[20] Rafael Microelectronics inc. *R820T High Performance Low Power Advanced Digital TV Silicon Tuner Datasheet*. Rafael Microelectronics inc., Taiwan, 10 2011.

[21] Realtek Semiconductor Corp. *RTL2832U DVB-T COFDM Demodulator+USB2.0 datasheet*. Realtek Semiconductor Corp., Taiwan, 1.4 edition, 11 2010.

[22] Rohde & Schwarz. *SMA100B RF AND MICROWAVE SIGNAL GENERATOR - Specifications*. Rohde & Schwarz, Munich, Germany, 8.02 edition, 1 2025. URL https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/pdm/cl_

brochures_and_datasheets/specifications/5215_1018_22/SMA100B_specs_en_5215-1018-22_v0802.pdf.

[23] M. Sadiku and C. Akujuobi. Software-defined radio: a brief overview. *IEEE Potentials*, 23(4):14–15, 2004. doi: 10.1109/MP.2004.1343223.

[24] Siglent Technologies co. *SSA3000X Series Spectrum Analyzer*. Siglent Technologies co., Shenzhen, China. URL https://siglent.co.uk/pdf/SIGLENT-SSA3021X-SPECTRUM-ANALYSER-Datasheet.pdf.

[25] D. Sinha, A. K. Verma, and S. Kumar. Software defined radio: Operation, challenges and possible solutions. In *2016 10th International Conference on Intelligent Systems and Control (ISCO)*, pages 1–5, 2016. doi: 10.1109/ISCO.2016.7727079.

[26] Skyworks Solutions, Inc. *Si5351A/B/C-B I2C-PROGRAMMABLE ANY-FREQUENCY CMOS CLOCK GENERATOR + VCXO*. Skyworks Solutions Inc., Woburn, Massachusetts, USA, 8 2021. URL https://www.skyworksinc.com/-/media/Skyworks/SL/documents/public/data-sheets/Si5351-B.pdf.

[27] R. Stallman. What is free software? URL https://www.fsf.org/about/what-is-free-software.

[28] T. Stübler. Hackrf toolbox with spectrum analyzer, 2015. URL https://www.mathworks.com/matlabcentral/fileexchange/55537-hackrf-toolbox-with-spectrum-analyzer.

[29] T Object Management Group Inc. Uml, 2025. URL https://www.uml.org/.

[30] Tektronix inc. *Arbitrary Waveform Generators - AWG70000B Series Datasheet*. Tektronix inc., Beaverton, Oregon, USA, 76w-61412-26 edition, 3 2024. URL https://download.tek.com/datasheet/AWG70000B-Datasheet_EN-US_76W-61412-26.pdf.

[31] Teledyne LeCroy inc. *WaveMaster 8000HD Datasheet Series Datasheet*. Teledyne LeCroy inc., Chestnut Ridge, New York, USA, wavemaster8000hd-ds-18oct24 edition, 10 2024. URL https://cdn.teledynelecroy.com/files/pdf/wavemaster8000hd-datasheet.pdf.

[32] The MathWorks Inc. Call c/c++ mex functions from matlab, 2025. URL https://it.mathworks.com/help/matlab/call-mex-file-functions.html.

[33] The MathWorks Inc. qammod - quadrature amplitude modulation (qam) documentation, 2025. URL https://it.mathworks.com/help/comm/ref/qammod.html.

[34] The MathWorks Inc. pskmod - modulate signal using m-psk method documentation, 2025. URL https://it.mathworks.com/help/comm/ref/pskmod.html.

[35] The MathWorks Inc. Rtl-sdr support from communications toolbox, 2025. URL https://it.mathworks.com/hardware-support/rtl-sdr.html.

[36] The Osmocom project. The rtl-sdr project, 2025. URL https://osmocom.org/projects/rtl-sdr/wiki/Rtl-sdr.

[37] K. Tomlinson. Prbs (pseudo-random binary sequence), 2015. URL https://blog.kurttomlinson.com/posts/prbs-pseudo-random-binary-sequence.

[38] M. Wang. Wimax physical layer: Specifications overview and performance evaluation. In *2011 IEEE Consumer Communications and Networking Conference (CCNC)*, pages 10–12, 2011. doi: 10.1109/CCNC.2011.5766338.

[39] S. Wiedenbeck, V. Ramalingam, S. Sarasamma, and C. L. Corritore. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. *Interacting with Computers*, 11(3):255–282, 1999. doi: 10.1016/S0953-5438(98)00029-0.

A | Tabulated measured data

Power measured and used to obtain graph 3.11 in chapter 3. These values represent the maximum power output of the HackRF One SDR measured using a single tone signal, setting the IF gain to the maximum (47dB) and enabling the final amplifier stage.

Frequency	Power
1MHz	6.7dBm
26MHz	15.7dBm
100MHz	8.8dBm
230MHz	6.1dBm
500MHz	9.1dBm
750MHz	1.0dBm
1000MHz	2.5dBm
1300MHz	-6.7dBm
1600MHz	-14.8dBm
1800MHz	-8.0dBm
2169MHz	-7.0dBm
2170MHz	5.3dBm
2739MHz	10.5dBm
2741MHz	-3.0dBm
3999MHz	-13.0dBm
4100MHz	-13.0dBm
6000MHz	-52.0dBm

Table A.1: HackRF maximum power output over its frequency range.

Due to its extensive length, the Nooelec Nesdr Smart V5 calibration table discussed in sections 2.5 and 2.9 is only available in .mat or .xls file on the project repositoy and can be found at the link: `https://github.com/Vogs27/CotSA`.
The following table shows data measured during the test described in section 5.2. The

first column contains the HackRF gain used, the second column the power measured at the input of the DUT, the third column the power at the ouput of the DUT.

Table A.2: P_{1dB} test (section 5.2) measured values.

HackRF Gain	DUT power in	DUT power out
0dB	-40.37dBm	18.79dBm
1dB	-39.28dBm	17.79dBm
2dB	-38.30dBm	16.85dBm
3dB	-37.43dBm	15.84dBm
4dB	-36.39dBm	14.94dBm
5dB	-35.52dBm	13.99dBm
6dB	-34.58dBm	12.99dBm
7dB	-33.56dBm	12.01dBm
8dB	-32.70dBm	11.15dBm
9dB	-31.77dBm	10.17dBm
10dB	-30.98dBm	-9.23dBm
11dB	-30.10dBm	-8.31dBm
12dB	-29.15dBm	-7.36dBm
13dB	-28.34dBm	-6.44dBm
14dB	-27.49dBm	-5.53dBm
15dB	-26.53dBm	-4.56dBm
16dB	-24.27dBm	-2.27dBm
17dB	-23.12dBm	-1.14dBm
18dB	-22.11dBm	-0.17dBm
19dB	-21.18dBm	0.72dBm
20dB	-20.20dBm	1.69dBm
21dB	-19.30dBm	2.58dBm
22dB	-18.39dBm	3.51dBm
23dB	-17.41dBm	4.47dBm
24dB	-16.44dBm	5.43dBm
25dB	-15.57dBm	6.27dBm
26dB	-14.62dBm	7.21dBm
27dB	-13.70dBm	8.11dBm
28dB	-12.81dBm	8.98dBm
29dB	-11.85dBm	9.87dBm
30dB	-10.96dBm	10.70dBm

HackRF Gain	DUT power in	DUT power out
31dB	-10.06dBm	11.51dBm
32dB	-9.10dBm	12.34dBm
33dB	-8.22dBm	13.06dBm
34dB	-7.29dBm	13.79dBm
35dB	-6.36dBm	14.48dBm
36dB	-5.44dBm	15.14dBm
37dB	-4.52dBm	15.76dBm
38dB	-3.60dBm	16.31dBm
39dB	-2.69dBm	16.73dBm
40dB	-1.76dBm	17.07dBm
41dB	-0.86dBm	17.34dBm
42dB	0.03dBm	17.57dBm
43dB	0.93dBm	17.77dBm
44dB	1.82dBm	17.94dBm
45dB	2.68dBm	18.08dBm
46dB	3.51dBm	18.21dBm
47dB	4.21dBm	18.29dBm

List of Figures

Figure 1.2 courtesy of the TinySA project: E. Kaashoek. TinySA ULTRA technical description, 10 2023. URL: `https://tinysa.org/wiki/pmwiki.php?n=TinySA4.TechnicalDescription`.

Figures 3.3a and 3.3b courtesy of the HackRF project group: M. Ossmann. HackRF One block diagrams. URL: `https://hackrf.readthedocs.io/en/latest/hardware_components.html#block-diagrams`.

List of Tables

Acknowledgements

The reader will excuse the author if this paragraph is written in Italian.

Inevitabilmente e giustamente, al termine non solo di un lavoro impegnativo, ma anche di un intero percorso di studi, non posso esimermi dal ringraziare chi mi ha permesso di giungere fino a qui. Non seguirò un ordine di importanza, poiché non credo si possa fare una tale classifica.
Un primo sentito ringraziamento va al prof. Matteo Oldoni in qualità di mio relatore, per aver creduto in me e nel progetto che è stato portato avanti e di cui si può leggere nelle pagine precedenti, e per avermi offerto diverse opportunità di crescita culturale durante questi mesi di lavoro.
Un altrettanto sentito ringraziamento a tutte le persone con cui ho avuto l'opportunità di interagire e collaborare durante il tempo passato nel laboratorio del dipartimento, tra cui: Davide Scazzoli, Prof. Francesco Linsalata, Prof. Maurizio Magarini e i dottorandi che ho conosciuto e che, per motivi di spazio, non posso elencare.
Il vostro contributo, supporto e presenza hanno sicuramente avuto un forte impatto su questi mesi trascorsi assieme: qualsiasi sia il mio futuro e la mia carriera, porterò sempre con me un ricordo positivo e di sincera stima nei vostri confronti.
Un grande grazie alla mia famiglia, per aver creduto in me e avermi supportato anche, ma non solo, economicamente in questo lungo percorso.
Giulia, hai sacrificato parte di te e del tuo tempo per rimanere al mio fianco durante questa avventura: non so se esista un modo per ringraziarti adeguatamente.
Vorrei, infine, dedicare questo lavoro a colui che con paterna dedizione mi ha insegnato l'amore per la curiosità; nella speranza che la lettura di questo scritto possa un minimo ispirare ad altri lo stesso amore: ovunque tu sia, grazie.

Open source software and hardware acknowledgment

This work would not have been possible without the work of many people contributing to the FOSS community and believing in free and open software and hardware. Clearly, the list cannot be exhaustive, so it makes sense to mention the groups who contributed to the hardware and software used in this project, instead of mentioning each person: the RTL-SDR community, with particular consideration to the Osmocomm group, and the GratScott Gadgets community and developers.
As a firm believer in FLOS principles, especially in an education context, the code, software and material produced with this thesis are available for download at `https://github.com/Vogs27/CotSA`, with the only condition to maintain the same licensing and attribution.