Lukas Kaltenleitner, BSc

Software Defined Radio based Nuclear Quadrupole Resonance Spectrometer

MASTER'S THESIS

to achieve the university degree of
Diplom-Ingenieur

Master's degree programme:
Biomedical Engineering

submitted to
Graz University of Technology

Supervisor
Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Hermann Scharfetter
Institute of Biomedical Imaging

Graz, February 2022

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master's thesis.

Lukas Kaltenleitner, BSc

Date, Signature

Abstract

Nuclear Quadrupole Resonance (NQR) spectroscopy is an Nuclear Magnetic Resonance (NMR) spectroscopy-related investigation method of the spin transitions of quadrupole nuclei. The samples contain nuclei with a quadrupole moment (spin quantum number $I > \frac{1}{2}$). These are introduced into a coil, excited at different frequencies with RF-pulses and the reinduced signals are measured to generate a spectrum. Due to the low amplitude, the induced voltages are difficult to measure, and require a sensitive receiver unit and adequate impedance matching to the measuring unit at any frequency.
An NQR spectrometer has basically the same hardware components as a NMR system (so the transmit and receive path of the spectrometers are identical), except the magnet for the static magnetic field which is not required. The hardware of an NQR system has to deal with radio frequencies in the range from a few MHz up to several hundreds of MHz, some special systems for nuclei with very high quadrupolar coupling constant up to the GHz-range.
Commercially available spectrometer systems are usually expensive and they are often difficult to adapt for special purposes.
New achievements in RF-technology make digitally programmable RF-transceivers known as Software Defined Radios (SDRs) affordable. SDRs make a big variety of RF-hardware digitally available and the majority of the signal processing is also done digitally. These SDRs often cover large frequency ranges and their design makes them almost natural candidates for use as spectrometer consoles. In this work, an implementation of an NQR system is introduced, which uses a 'LimeSDR' from 'Lime Microsystems' as a spectrometer console.

Key Words: Nuclear Quadrupole Resonance, Spectroscopy, LimeSDR, Sofware Defined Radio, Lime Microsystems

Kurzfassung

Die Kern-Quadrupol-Resonanz-Spektroskopie (NQR) ist eine der Kernspinresonanzspektroskopie (NMR) verwandte Untersuchungsmethode der Spinübergänge von sogenannten Quadrupolkernen.
Die Proben enthalten Kerne mit einem Quadrupolmoment (Spinquantenzahl $I > \frac{1}{2}$). Diese werden in eine Spule eingeführt, bei verschiedenen Frequenzen mit HF-Pulsen angeregt und die zurück induzierten Signale gemessen, um ein Spektrum zu erzeugen. Aufgrund der geringen Amplitude sind die induzierten Spannungen schwierig zu messen. Das fordert einerseits einen sensitiven Empfänger und andererseits eine angemessene Impedanzanpassung an die Messeinheit bei jeder applizierten Frequenz.
Ein NQR-Spektrometer hat im Grunde die gleichen Hardwarekomponenten wie ein NMR-System (der Sende- und Empfangsteil der Spektrometer ist also identisch), mit Ausnahme des Magneten für das statische Magnetfeld, welcher bei NQR nicht benötigt wird. Die Hardware eines NQR-Systems muss mit Frequenzen im Bereich von wenigen MHz bis zu mehreren hundert MHz zurechtkommen, Systeme für Kerne mit hoher quadrupolarer Kopplung sogar bis in den GHz-Bereich.
Kommerziell erhältliche Spektrometersysteme sind in der Regel teuer und lassen sich für spezielle Zwecke oft nur schwer anpassen.
Neue Errungenschaften in der HF-Technologie machen digital programmierbare HF-Transceiver, sogenannte Software Defined Radios (SDRs) und somit eine große Vielfalt an HF-Hardware digital verfügbar. Diese SDRs decken oft große Frequenzbereiche ab und ihr Design macht sie zu fast natürlichen Kandidaten für den Einsatz als Spektrometerkonsolen. In dieser Arbeit wird die Implementierung eines NQR-Systems vorgestellt, das ein 'LimeSDR' von 'Lime Microsystems' als Spektrometerkonsole verwendet.

Schlüsselwörter: Kernquadrupolresonanz, Spektroskopie, LimeSDR, Sofware Defined Radio, Lime Microsystems

Acknowledgements

First and foremost, I would like to thank my supervisor and mentor, Professor Hermann Scharfetter, for his patience and the outstanding enthusiasm he passed on to me. His teaching shaped me throughout my studies and made me a much better engineer than I could have ever been without his guidance. His commitment during the corona pandemic should also be highlighted, since he kept the laboratory open even during the most difficult period of my university career.

I would like to express a second and equally big thank to Andrin Doll. His project made this work possible in the first place. Thank you Andrin for always supporting me and answering all the questions that came up in this context.

My deepest thanks also go to all those who made my time at Graz University of Technology such a wonderful and educational time. So special thanks to Anna, Johanna, Gabriel, Christoph, David, Mario, Thomas, Bernhard and Stefan to mention just a few.

Without the trust, love and financing of my studies by my beloved parents Sabine and Bernhard, none of this would have been possible. Thanks mom and dad for all the support and for always listening to me when I was on the verge of despair. Another special thanks goes to my beloved sister Lisa, who also sacrificed many hours to support me. Through you I will always know where my home is.

Last but not least, I would like to thank Dominik and Martin, for any support during my studies.

Contents

List of Figures

List of Tables

List of Acronyms

NWA Network Analyzer
PA Power Amplifier
FID Free Induction Decay
SDR Software Defined Radio
NMR Nuclear Magnetic Resonance
NQR Nuclear Quadrupole Resonance
TM Tuning and Matching / Tune Match
TU Graz Graz University of Technology
KFU Graz University of Graz (Karl Franzens University)
LNA Low Noise Amplifier
LV low voltage
HV high voltage
SNR Signal to Noise Ratio
std standard deviation
RF Radio Frequency
RMS Root Mean Square
PCB Printed Circuit Board
TX Transmit
RX Receive
FFT Fast Fourier Transform
GPIO General Purpose Input Output
LO Local Oscillator
IF Intermediate Frequency
BB Base-Band
PLL Phase Locked Loop
VCO Voltage Controlled Oscillator
IQ In- phase and Quadrature-phase (complex signals)
EXOR Exclusive OR (Logic)
FWHM Full-Width-Half-Maximum
EFG Electric Field Gradient
FPGA Field Programmable Gate Array
FPRF Field Programmable Radio Frequency
ADC Analog to Digital Converter
DAC Digital to Analog Converter
TSP Transceiver Signal Processor

1 Introduction

Nuclear Quadrupole Resonance (NQR) spectroscopy is an Nuclear Magnetic Resonance (NMR) spectrocopy-related investigation method of the spin transitions of quadrupole nuclei. The samples contain nuclei with a quadrupole moment (spin quantum number $I > \frac{1}{2}$). These are introduced into a coil, excited at different frequencies with RF-pulses and the reinduced signals are measured to generate a spectrum. Due to the low amplitude, the induced voltages are difficult to measure, which requires a sensitive receiver unit and adequate impedance matching to the measuring unit at any frequency.
An NQR spectrometer has basically the same hardware components as an NMR system (so the transmit and receive path of the spectrometers are identical), except the magnet for the static magnetic field which is not required. The hardware of an NQR system has to deal with radio frequencies in the range from a few MHz up to several hundreds of MHz, some special systems for nuclei with very high quadrupolar coupling constant up to the GHz-range.
Commercially available spectrometer systems are usually expensive and they are often difficult to adapt for special purposes. One achievement in today's RF-technology are digitally programmable RF-transmitters and receivers, so-called Software Defined Radios (SDRs). These are usually supported through open source projects and platforms and offer an inexpensive option for a wide variety of RF applications. Various SDRs are designed as so-called transceivers, which can both send and receive. They usually cover wide frequency ranges from a few MHz to the single-digit GHz range. An SDR makes a wide variety of components, which are usually implemented as hardware parts, digitally available and the majority of the signal processing is also done digitally. For example, amplifiers and filter structures are standard elements of an SDR. The device used in this work is the 'LimeSDR' from 'Lime Microsystems', which is a full duplex Software Defined Radio that can be used as spectrometer console. Per default, this device can handle frequencies in the range of $100\,kHz$ to $3.8\,GHz$, whereas in the course of the NQR research carried out at the Institute of Biomedical Imaging, the lower frequency range of around $20 - 300\,MHz$ is dealt with in particular.

1.1 History, Inspiration and Motivation

Due to the high costs of commercially available spectrometers system and the fact that they are often not easily adaptable to specific needs, it is important to find affordable alternatives for research and development.

1.1.1 History of NQR Spectroscopy at Institute of Biomedical Imaging, Graz University of Technology

At the Institute of Biomedical Imaging a first custom-made NQR spectrometer was set up using a ZVL Vector Network Analyzer (Rhode and Schwarz) [27]. The entire system, built over several years, is known as 'CONCRADLE'. In the framework of a research project a commercial spectrometer (Scout, Tecmag) was purchased. In both cases, the price exceeded the two-digit thousand euro range for the console alone. Specific problems which arose during some special experiments could be fixed only with considerable effort or sometimes there was no solution because of limited access to some interior units of the commercial harware parts. On the one hand, with the 'CONCRADLE' system, the highly important phase-cycling could only be implemented very late and laboriously with external hardware and software (and here also only for parts of the total frequency range used). On the other hand, the 'SCOUT' system, unlike the network analyzer, does not allow to scan over the entire frequency spectrum in a single, fast scan, because the generic software does not support this mode of operation.
By comparison, the 'LimeSDR' costs just about 300 €, is programmable (i.e. can be configured for special purposes) and could be used without much hesitation, for example in an open students lab or a maker space to give interested students hands-on access to RF technology and NQR spectroscopy.

1.1.2 Inspiration, Related Work

The inspiration for this work came from the papers by Mr. Carl A. Michal [1] and Mr. Andrin Doll [2], whereby the essential idea of [2] and the open source files provided in his work [3] were used as the basis of this project.
In [1], the probably first successful attempt to realise an NMR spectrometer based on the 'LimeSDR' is described. The core achievement of this work is a synchronisation of a 'LimeSDR' with an 'Arduino Due' microcontroller which is used as a pulse programmer. The microcontroller triggers the SDR, which then performs the sequence, i.e. sends the RF-pulses and receives the resulting signals after excitation. A slightly different implementation was carried out in [2]. Here, in addition to the RF-pulses, GPIO signals are generated by the SDR, which can trigger other external components. For example, this GPIO signal can be used

as a gate signal for the power amplifier (PA) used.
In both projects ([1] and [2]), however, the Field Programmable Gate Array (FPGA) firmware of the 'LimeSDR' had to be modified and a C++ routine had to be created to implement the necessary function.

Both publications ([1] and [2]), led to fully functional NMR consoles. Since in [2] the hardware effort is lower (no 'Arduino' necessary for sequence generation) and the handling seemed to be easier, this implementation was chosen as the basis for this thesis.

1.1.3 Motivation

The idea to create a cost-effective alternative for NQR spectroscopy as well as to give students the opportunity to gain experience in NMR-related RF-technology is the driving force behind this project.

1.2 Physical Basics of NMR and NQR Spectroscopy

This section should introduce the physical basics of NMR spectroscopy and form a transition to NQR Spectroscopy. It is intended to highlight both the similarities in the measurement method and the physical differences.
First, a consideration should be introduced in order to compare the two types of spectroscopy. A detailed explanation follows in the respective subsection.

Consideration:
In both methods, the same hardware can be used, with the difference that NMR experiments also require a magnet for the static magnetic field. Figure 1 shows a schematic of an NMR system.

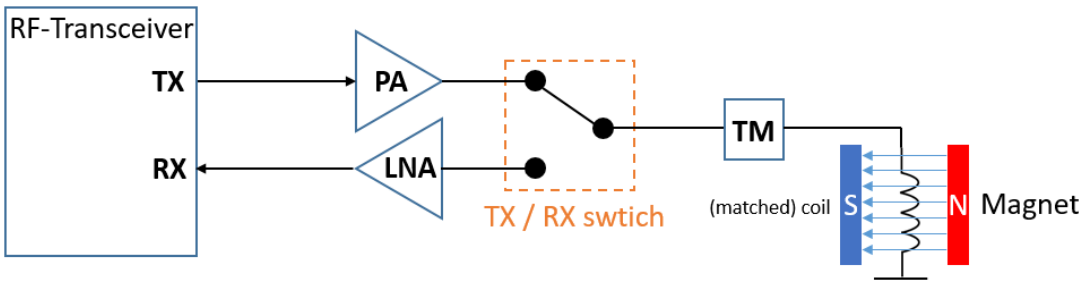

Figure 1: Schematic of an NMR system.

The static magnetic field is produced by the magnet (see figure 1) and the pulsed RF-field is produced by a RF-transceiver. The RF-pulses are transmitted to a matched coil, where the sample is located. After that, the system is switched to RX mode, where the induced signals of the sample are measured.

Now there is the question why this field is necessary and why NQR experiments do not need it.
For the initial situation, it is necessary to create a preferred direction for the orientation of the magnetic moments of the atomic nuclei. This is the case for both spectroscopy methods. Once this has been achieved, the macroscopic magnetization of the different spin populations can be tilted by applying pulsed RF-fields with a certain frequency to the coil where the sample is located. This happens in addition to the static magnetic field when talking about NMR. Here it is important that the selected frequency of the RF-pulses fulfils the resonance condition, i.e. that the excitation frequency corresponds to the energy difference between two spin energy levels. After excitation, signals produced by the sample can be detected at the receiver at the same frequency.

1.2.1 NMR Spectroscopy

Certain elements of the periodic table have an atomic nuclear spin. The particles in an atomic nucleus are protons and neutrons, each of which has its own spin. Atoms that have an odd number of nuclear particles have a natural nuclear spin. If the number of nuclear particles is even, the atom is magnetically neutral. Conversely, an atomic nucleus with a nuclear spin is always magnetic. About two thirds of all elements found in nature have a nuclear spin. [5]

The spin is a result of the intrinsic angular momentum $\vec{P}$ which is in measurements appears as a quantized property. The value of the total angular momentum depends on the spin quantum number I and the reduced Planck's constant $\hbar$. It is calculated as follows:

$$|\vec{P}| = \sqrt{I(I+1)\hbar} \tag{1}$$

Associated with $\vec{P}$ is the magnetic momentum $\vec{\mu}$. The magnetic moment $\vec{\mu}$ is proportional to $\vec{P}$ by the magnetogyric ratio γ.

$$\vec{\mu} = \gamma\vec{P} \tag{2}$$

The magnetogyric ratio γ is specific for each nucleus. [6]

An NMR sample, or as in magnetic resonance imaging (MRI) the human body, contains a very large number of atoms whose nuclei have a nuclear spin, which in turn depends on the element. In a field-free space (no external magnetic field), the spins in matter are arranged completely randomly and do not lead to a macroscopic net magnetization. In presence of a static magnetic field at room temperature, there will still be a quite random orientation of the spins, but according to statistical thermodynamics there will be a slight preference for an orientation along the magnetic field. This results in a macroscopic magnetization. As the spin population also has a macroscopic angular momentum, the whole magnetization behaves like a spinning top and starts precessing about the magnetic field. The build-up of the macroscopic magnetization is mediated by irregular, thermally driven magnetic interaction (magnetic forces) between the individual spins which finally yields a slight skew of the random distribution towards the external magnetic field direction.

Figure 2 shows the spins orientation in a field free space and with an external magnetic field. M indicates the (small) macroscopic sum magnetization. The orientation of the spins should be indicated by the denser upward arrangement in (b).

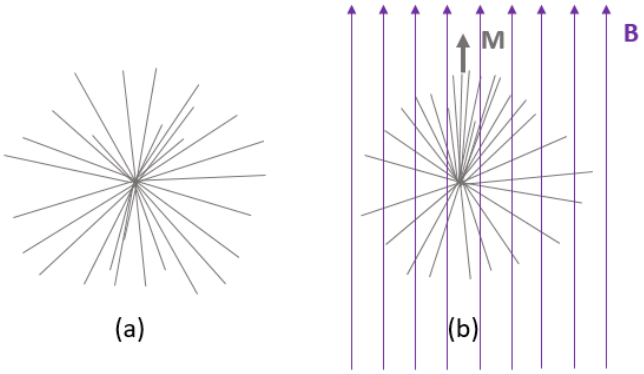

Figure 2: Illustration of the spins orientation in a) a field free space and b) when an external magnetic field $\vec{B}$. The idea of the representation was taken from [8], initially available in [7].

The conical movement of the net magnetization is called precession. An important piece of information is the frequency at which this motion is occurring. It depends on the gyromagnetic ratio γ and the applied static magnetic field B_0.

$$\omega = \gamma B_0 \quad , where\ \omega = 2\pi f \tag{3}$$

This frequency is called the Larmor frequency and can be denoted as:

$$f_{Larmor} = \frac{\gamma}{2\pi} B_0 \tag{4}$$

The above shown illustration (figure 2 b) can be drawn into a coordinate system. This can be seen in figure 3.

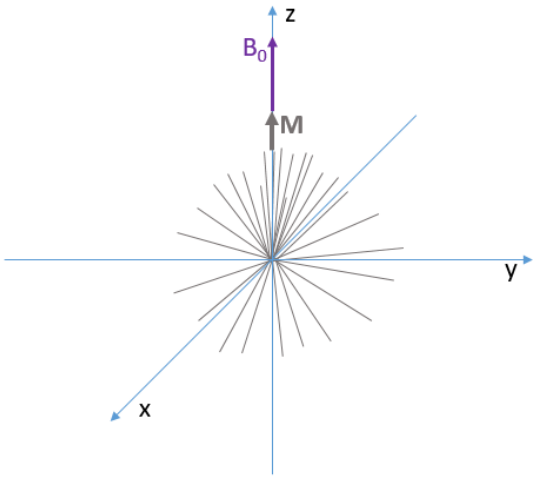

Figure 3: Spin orientations with a present static mangnetic field in a coordinate system.

As briefly mentioned in the consideration above, an RF-pulse which oscillates with the Larmor frequency can force the sum magnetization M to flip in the xy-plane. Therefore, the pulsed field (usually denoted as B_1) must be normal to the static magnetisation. A tuned and matched RF-coil is used as transmit and receive coil (see figure 1) to apply the RF-pulses and to receive the emmitted radio waves from the nuclei during relaxation. If the flip angle is exactly 90°, the maximum induction into the coil will take place.
Figure 4 shows the effect of an RF-pulse oscillating with the Larmor frequency additionally applied to the static magnetic field.

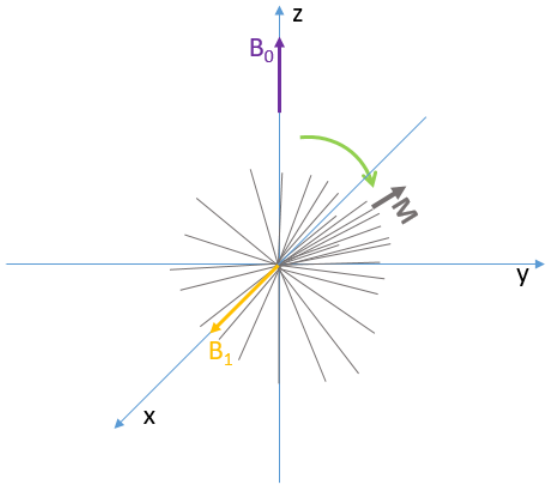

Figure 4: Result of an RF-pulse applied additionally to the static magnetic field.

After excitation, there are two dominant effects that cause the magnetisation in the xy-plane to decay and drive it back in the initial direction (z-direction). These two effects are called transverse (T_2) and longitudinal (T_1) relaxation.
The transverse relaxation is also called spin-spin relaxation. After excitation, the spins of the sum magnetisation are in phase, but due to mutual influence they get out of phase. In addition there is also dephasing due to external field inhomogeneities, which is described by the time constant T_2^*. A totally random phase of the spins means no signal measurable in the xy-plane.
The following figures should illustrate the dephasing of the spins. For simplicity, the spin ball has been reduced to spins oriented in the preferred direction. The coordinate system rotates with the Larmor frequency ω_{Larmor} (rotating frame). Figure 5 shows the dephasing of the spins due to T_2 relaxation.

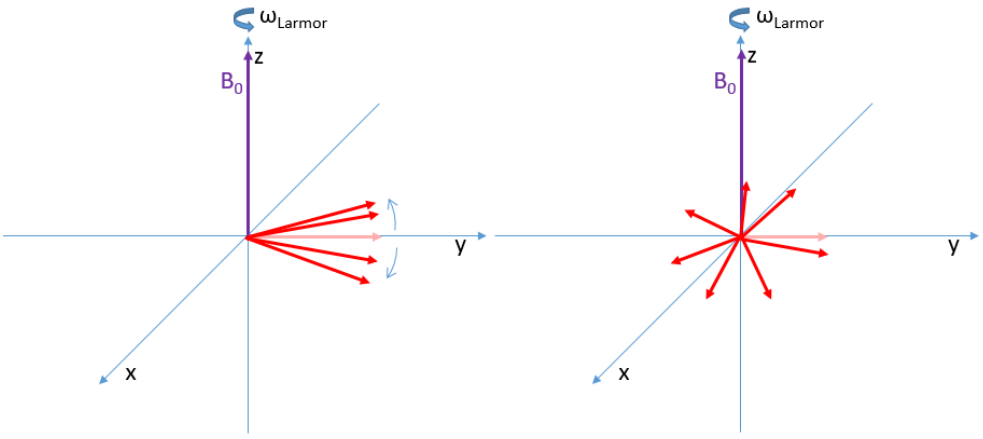

Figure 5: Dephasing of the spins due to T_2 relaxation.

Figure 6 shows the decay of the magnetization in xy-direction due to T_2 and T_2^* relaxation.

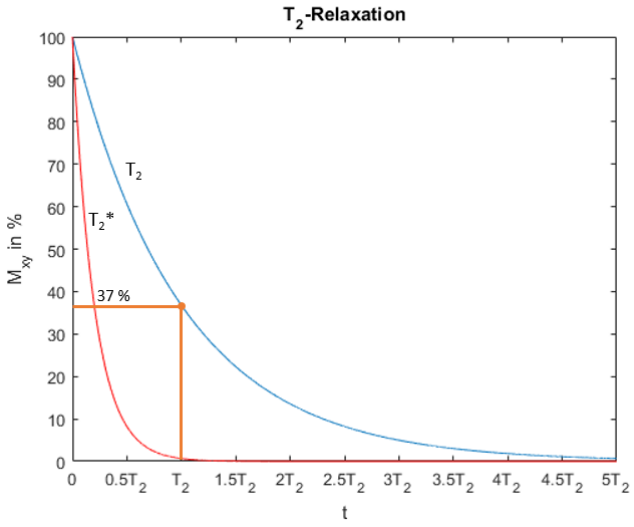

Figure 6: Decay of the magnetization in xy-direction due to T_2 and T_2^* relaxation.

The T_2 constant is defined as the time, where the magnetization M_{xy} in the xy-plane has dropped to 37% of the maximum.
The T_2 relaxation can be described with a exponential function as follows:

$$M_{xy}(t) = M_0 \cdot e^{-\frac{t}{T_2}} \tag{5}$$

where M_0 is the initial magnetization due to B_0.

The second effect, the T_1 relaxation is also called spin-lattice relaxation and it describes the relaxation back into z-direction.
Figure 7 shows the T_1 relaxation back into z-direction.

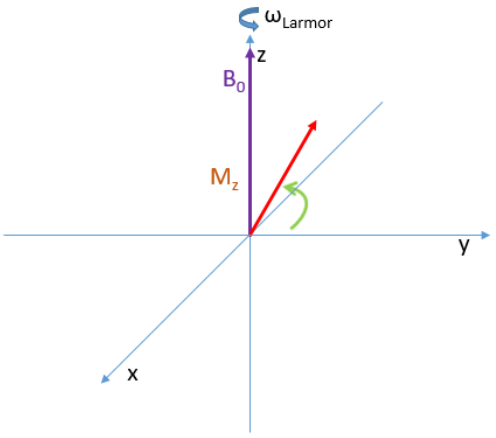

Figure 7: T_1 relaxation.

Figure 8 shows the return of the magnetization M_z in z-direction due to T_1 relaxation.

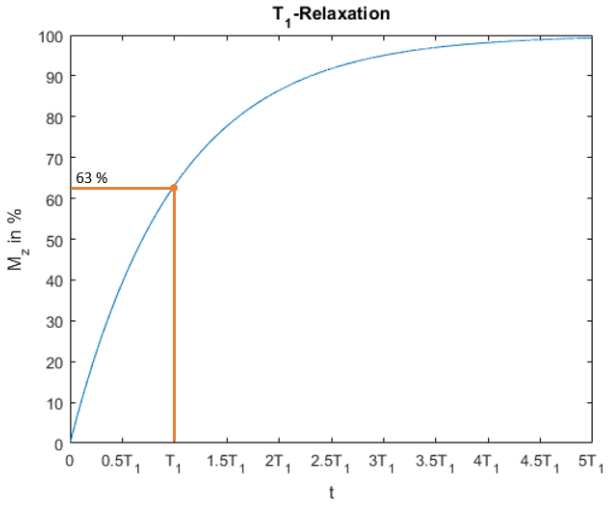

Figure 8: T_1 based return of the magnetization M_z in z-direction.

The T_1 constant is defined as the time, where the magnetization M_z in z-direction has reached 63% of the initial magnetization.
The T_1 relaxation can be described with a exponential function as follows:

$$M_z(t) = M_0 \cdot (1 - e^{-\frac{t}{T_1}}) \tag{6}$$

where M_0 is the initial magnetization due to B_0.

The flipping of the spins and the measurement of the signal after a certain time is called a pulse sequence. The two most widespread pulse sequences will now be explained. A single 90° pulse is called Free Induction Decay (FID), because the magnetization decreases exponentially in the xy-direction. Figure 9 shows the FID sequence. The decay of the FID is proportional to T_2^*. (Note: the excitation pulse and the FID signal are not proportional)

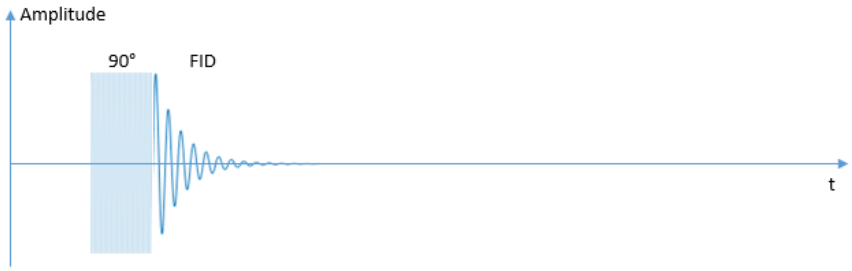

Figure 9: FID sequence.

Since some substances have a short T_2^* time, a sequence which refocuses the spins is often used. Therefore a second pulse must be applied. The second pulse usually has twice the length of the first one and therefore flips the spins by 180°. Due to the 180° pulse, the slower spins pass over the fast spins (they are in advance), and hence the different members of the population run together again. The rephasing is illustrated in figure 10.

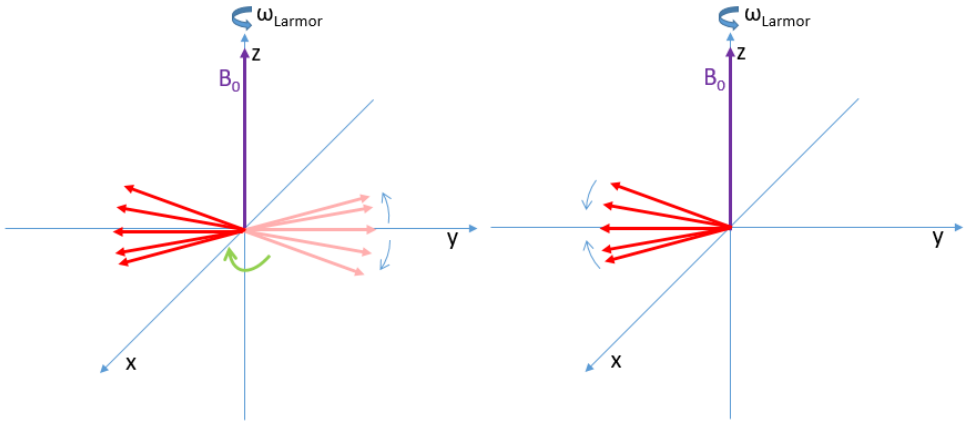

Figure 10: Rephasing of the spins with a 180° pulse.

Figure 11 shows the Spin-Echo Sequence. (Note: the excitation pulse and the Echo signal are not proportional)

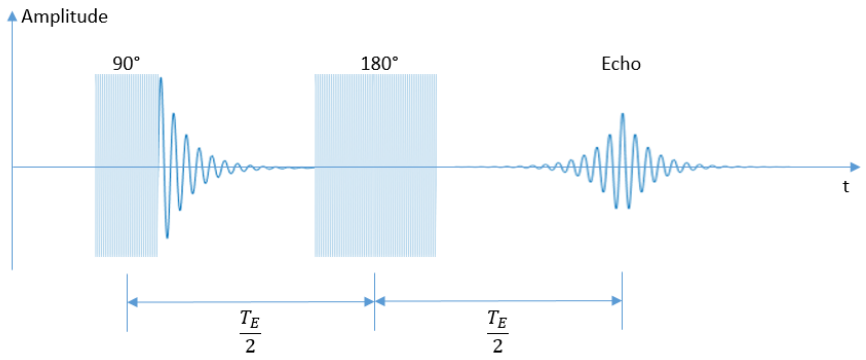

Figure 11: Spin-Echo sequence.

The time between the center of the 90° pulse and the peak of the resulting echo is called echo time (T_E). The echo time is determined by the distance of the pulses to each other.
The Spin-Echo has the advantage, that due to the 180° pulse, the decay of the peak signal is proportional to T_2.

1.2.2 NQR Spectroscopy

The above given introduction into the topic of NMR spectroscopy, should now be be associated with NQR spectroscopy. At the atomic level, nuclei that have

a quadrupole moment are distinguished by the spin quantum number from those nuclei without a quadrupole moment. Nuclei that have a spin quantum number $I > \frac{1}{2}$ have an electrical quadrupole moment.
The big difference now is: An atomic nucleus without a quadrupole moment has a homogeneous charge distribution at the nucleus and therefore interacts with a surrounding (homogeneous) static magnetic field (magnetic moments of nucleis align into the preferred direction and the spins starts precessing). Nuclei with a quadrupole moment do not have a homogeneous charge distribution at the nucleus and they still interact with magnetic fields unless their spin is 0. You can also do standard NMR experiments with quadrupolar nuclei. However, both magnetic and electric interaction are superimposed. In our nuclei the electric component by far exceeds the magnetic one, so that NQR gives higher transition frequencies. Quadrupole nuclei, however, do interact with inhomogeneous electric fields. These fields are automatically generated from the atoms by binding electrons moving around the nucleus. Extremely strong electric fields prevail in the vicinity of a (quadrupole) nucleus. Due to the presence of these fields, the spins and therefore also the magnetic moments of the nuclei automatically tend to align themselves in a certain preferred direction which in case of pure NQR is determined by the main axis of the electric field gradient (EFG) at the site of the nucleus. Many nuclei in a sample therefore have a natural preferred direction (crystals can also be grown in certain processes which then have a strong preferred direction). At TU Graz, mainly crystalline powders are used as samples. With a powder, about 30% of the nuclei's signal is lost since some of the spins are not oriented in the preferred direction ([25] on page 86).

This in short, is the reason why there is no need of an external force (like the static magnetic field in NMR) to align the magnetic moment of quadrupole nuclei.

As with NMR experiments, a flip of the preferred direction can also be achieved with NQR by pulsed RF-fields. This means that the pulse sequences work out analogously. However, there is a significant difference here. To achieve a flip in NQR, alternating fields are needed. In contrast, NMR requires rotating fields for excitation. The RF-pulses in a coil basically generate alternating fields. This raises another question, namely why NMR works at all with the same setup.
In mathematical terms, alternating fields can be decomposed into two opposing rotating fields. In the case of NMR, only one of the two rotating fields fulfils the Larmor condition. Strictly speaking, NMR only works with half the field efficiency if you compare it directly with NQR using a solenoit RF-coil.

1.3 NQR System

1.3.1 NQR Hardware

This section will give an overview of the hardware components used in an typical NQR spectrometer. Figure 12 shows the hardware components of an NQR system.

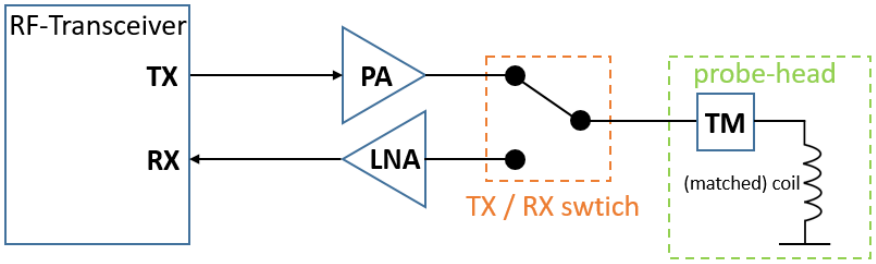

Figure 12: Hardware components of an NQR spectrometer.

The RF-transceiver is responsible for generating the RF-pulses and for receiving the re-induced signals from the sample coil. The power amplifier (PA) is used to amplify the RF-pulses to achieve the power necessary to excite the nuclei. The TM block represents a Tuning- and Matching-Network, which is used to tune and match the coil to 50 Ω at a certain target frequency. Note that the coil is built in combination with the matching-network (as one hardware element) and called probe-head. The matching-network is used during the RX phase to ensure adequate impedance matching of the coil at each excitation frequency. The signals in the RX phase are amplified by a Low Noise Amplifier (LNA) which is connected to the transceiver's RX port.

1.3.2 Working Principle

In transmit mode (TX path of the spectrometer is connected to the coil via the TX / RX switch) the RF-transceiver generates a pulsed RF-signal, which is amplified by the PA. This high power RF-pulse is then transmitted to a matched coil, in which the sample is located. A moment after the the excitation with the TX pulse, the system is switched into RX mode and the re-induced signal first passes the matching-network. It is then amplified with the LNA and digitized at the RX port of the transceiver. This procedure is typically repeated a few hundred to a few thousand times, since a sufficiently high averaging is necessary to reduce the noise in the measurement data. The averaged data can then be further processed digitally on a PC. The FFT (Fast Fourier Transformation) is usually applied to the data, since the data in the frequency range lead to the characteristic spectra and, thus, a very clear representation of the frequency components is created.

1.4 Software Defined Radio

Software Defined Radio (SDR) refers to RF-transmitters and receivers that make a large part of RF components that are otherwise realised in hardware digitally available. Typically, a large part of the signal processing is also implemented digitally [4]. These devices are very popular with radio amateurs and in communications technology in general, as they offer many possibilities in just one device and the function of the device can be changed very quickly by changing the software. They are also used in research and development, for example, SDRs are installed on satellites used for research purposes in order to carry out telecommunications tests.
Very well known SDRs are for example the 'ADALM PLUTO' (Analog Devices Inc), the 'HackRF One' (Great Scott Gadgets) or the 'LimeSDR' (Lime Microsystems) used in this work.
The primary difference between SDRs is often the available frequency range or the mode of data transmission. For example, the 'ADALM PLUTO' would not be applicable for the frequencies needed here, as it operates from $325\,MHz$ to about $3.8\,GHz$. An important difference between the 'LimeSDR' and the 'Hack RF One' is that the 'LimeSDR' works with full-duplex and therefore has separate TX and RX ports, whereas the 'Hack RF One' has only one antenna port and is therefore only half-duplex capable.
There are various ways to program an SDR. A frequently used software is 'GNURadio', which offers function blocks for various SDRs. GNURadio' offers a graphical interface to the control of SDRs with a flow graph. In addition, 'C', 'C++' or 'Python' APIs are often available to implement various routines.
A widespread functional principle that SDRs and also mobile phones use is the principle of a direct mixer according to the I/Q method [4]. This principle is also used by the 'LimeSDR' and will be explained in more detail later.

1.4.1 LimeSDR

The LimeSDR (USB Type A) is a combination of a USB 3.0 interface, a FPGA and the LMS7002M Field Programmable RF Transceiver (FPRF) IC. The entire system on the PCB, forms the so-called SDR. The core of the system is the LMS7002M IC (equipped with IQ mixers, filters, amplifiers etc.), which is responsible for synthesising and receiving the radio frequency. The FPGA used is an Altera Cyclone IV FPGA, which forms the channel to the PC with the help of the USB 3.0 interface. A more detailed explanation is given in the methods section.

1.5 Initial situation and requirements of this work

1.5.1 Initial situation

The software provided in the open source project [3] contains a C++ routine that communicates with the FPGA of the SDR via the USB 3.0 interface, i.e. sets the necessary registers using C++ commands. The FPGA then makes the necessary settings on the LMS7002M FPRF chip. This routine is responsible for all necessary settings such as the sample rate, TX gain, RX gain, etc. and handles the entire transmit and receive process. To allow easy scripting of the sequences, the project includes a Python file (name: limr.py) which passes all settings (also writable in a python file) to the C++ routine, which must be compiled beforehand (see appendix figure 117). Thus, the executable routine is called by the Python script and the set-up experiment is carried out. The time domain data acquired at the RX port (which is complex due to IQ sampling) are then stored (data acquired at a certain frequency is summed up with respect to the chosen number of averages) in an hdf5 file and can be processed further from there. There is also the possibility of phase cycling. The TX phase can be set and the RX phase can be adjusted in post-processing using the stored data. The interaction of the PC with the LimeSDR can be seen in figure 13. Figure 13 is a slightly modified version of the figure in [2].

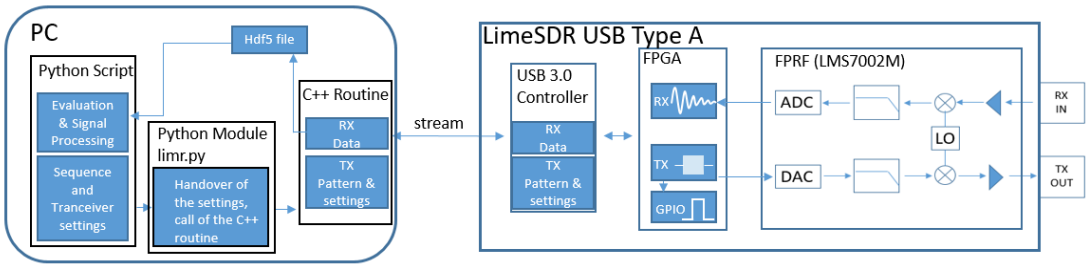

Figure 13: Illustration of the interaction between the PC and the LimeSDR.

Both, the C++ routine and the python script are running on a Linux machine (used Ubuntu 20.04.2) with a USB port $\geq$ 3.0.

1.5.2 Requirements of this work

The aim of this work is to explore the usage of the project [2] and the associated files provided in [3] for NQR spectroscopy. In this work, the application of the LimeSDR in NQR Spectroscopy should be experimentally confirmed by building up a system with the required hardware and by creating and testing self designed pulse sequences.

2 Methods

2.1 Important RF principles and assemblies

2.1.1 Phase Locked Loop (PLL)

Phase Locked Loops (PLLs) are used for signal generation because they can generate higher and lower frequency signals from a reference signal with appropriate dividing and multiplication factors. PLLs can cover a wide range of frequencies and are ideal for all devices that need to be able to generate signals of different frequencies. Many devices work with low-frequency base-band signals, which are mixed with a signal of higher frequency (usually called local oscillator (LO)) to shift it into a high-frequency band for transmission. This local oscillator frequency is usually generated by a PLL.

Figure 14 shows a schematic of a phase locked loop.

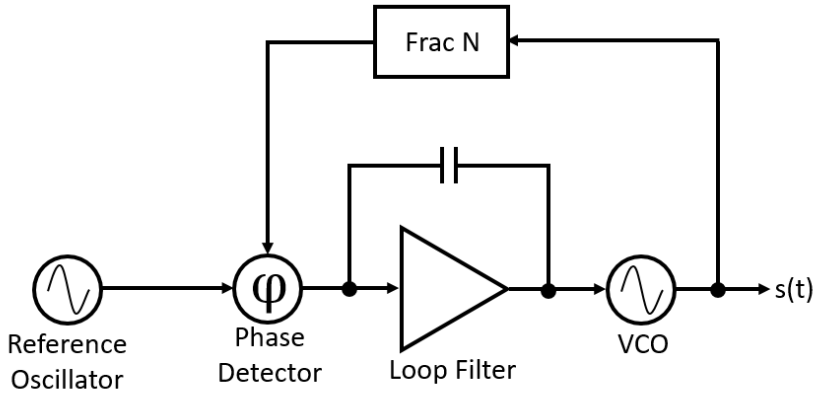

Figure 14: Schematic of a Phase Locked Loop.

The basic principle of a PLL is best explained using figure 14. The reference oscillator (usually a quartz oscillator) has a fixed reference frequency whose phase is compared with the phase of the PLL signal by a phase detector. The phase comparator can be imagined digitally as an exclusive or (EXOR) gate, which is only high when the phases are different. This signal is then filtered with the help of a loop filter and thus converted into an input signal for the following Voltage Controlled Oscillator (VCO). A VCO has the ability to vary its frequency depending on the voltage applied. This is usually done by a varactor diode, which changes the junction capacitance depending on the applied voltage. The output signal of the VCO is then fed to the phase comparator via a frequency divider. The frequency divider makes it possible for the PLL to 'lock' to different division ratios (hence the 'locked' in the PLL's name). This arrangement makes

it possible to generate different output signals (according to the divider ratio and the reference frequency) [9].

2.1.2 RF-Mixer

RF mixers are components often used in communication technology. They can mix a low-frequency base-band signal (BB), often also called intermediate frequency (IF), with a local oscillator (LO) and thus generate the high-frequency signal (RF) to be transmitted and are also capable of mixing a high frequency (RF) signal down to the base-band (IF). This principle is used, for example, in broadcasting. The base-band signal carries the information (e.g. an audio signal which was modulated onto the base-band signal) and is mixed with the high-frequency carrier (which is the local oscillator in figure 15). Each receiver (e.g. a radio) can then first mix the RF signal down to the base-band and then demodulate the received signal accordingly and can thus further process the base-band signal.

Figure 15 shows the schematic of an RF-mixer. Picture a) shows a so-called upward mixer (typically installed in the transmitter), which produces the RF as an output, picture b) shows a downward mixer (typically installed in the receiver) which mixes the received RF-signal down to the base-band so its output is the IF signal.

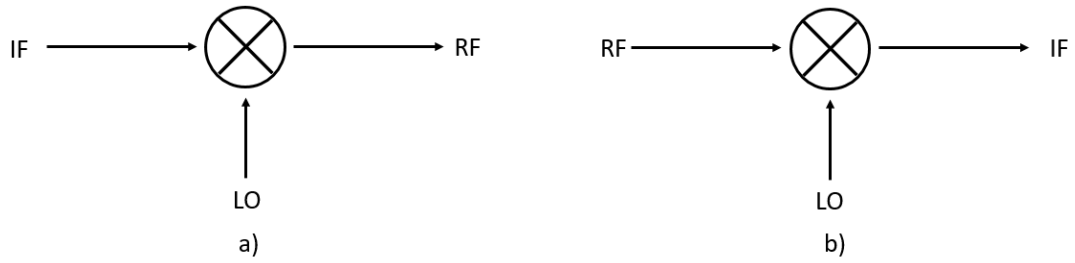

Figure 15: RF mixer schematic.

There are different types of mixers, the most common being multiplicative mixers. Mathematically, the multiplicative upward mixer can be described as follows.

Let the IF and the LO signal be cosine signals:

$$IF = cos(2\pi f_{IF} t) = \frac{e^{j2\pi f_{IF} t} + e^{-j2\pi f_{IF} t}}{2} \tag{7}$$

and:

$$LO = cos(2\pi f_{LO} t) = \frac{e^{j2\pi f_{LO} t} + e^{-j2\pi f_{LO} t}}{2} \tag{8}$$

The two input signals are then multiplied by the mixer:

$$RF = \frac{e^{j2\pi f_{IF}t} + e^{-j2\pi f_{IF}t}}{2} \cdot \frac{e^{j2\pi f_{LO}t} + e^{-j2\pi f_{LO}t}}{2} \tag{9}$$

This can be further written as:

$$RF = \frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} + e^{j2\pi(f_{IF}-f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}-f_{LO})t}) \tag{10}$$

And brought back to cosine relationships:

$$RF = \frac{1}{2}(cos(2\pi(f_{IF} + f_{LO})t) + cos(2\pi(f_{IF} - f_{LO})t)) \tag{11}$$

It can be seen (equation 11), that the product of a multiplicative mixer is always a lower and an upper side-band, which is either the sum or the difference of the IF and the LO frequency. Typically, only one side-band is transmitted, which is selected with a bandpass filter.

The downward path works analogously (it mixes the RF signal with typically the same LO), producing one signal containing $|f_{LO} - f_{RF}|$ and $|f_{LO} + f_{RF}|$ (as there exists no negative frequency). Typically, an intermediate frequency filter is then used to select the lower band, i.e. $IF = |f_{LO} - f_{RF}|$.

2.1.3 IQ-Mixer and Single Side-Band Modulation

A method often used for signal synthesis (and also for receiving or digitising analogue signals) is that of a complex mixer. Complex mixers, or IQ mixers, use a signal track I and a signal track Q, where I stands for in-phase and Q for quadrature phase (Q is shifted by 90° from I). The term complex mixer comes from the mathematical description of signals and is based on the fact that in the complex plane the real and imaginary parts are shifted by 90°. Compared to the simple multiplicative mixer discussed above, this method offers some advantages. First, one of the side-bands produced by the multiplicative mixer is usually unwanted and must be filtered. With the IQ mixer one of the two side-bands can be suppressed and this problem does not arise.
On the receiver side, complex signals also offer some advantages. For example, the phase information can be obtained through complex sampling and it is theoretically possible to work with a sampling rate below Shannon (Shannon stipulates that a signal must be sampled with at least twice the maximum frequency occurring in the signal $f_s \geq 2f_{max}$ in order to reconstruct the frequency; with complex

sampling, a sample rate equal to the maximum frequency occurring in the signal is theoretically sufficient).

Figure 16 shows the schematic of an IQ mixer (transmitter and receiver case).

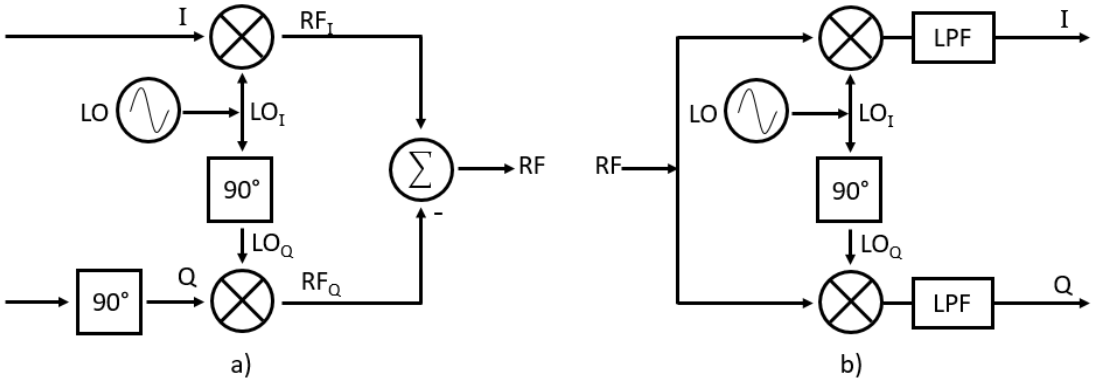

Figure 16: a: IQ mixer schematic with upper side-band selection (transmitter), b: IQ mixer as receiver. The structure was taken from [26] but redrawn and supplemented.

Usually, the I and Q components are of low frequency lying in the base-band which are then mixed with the local oscillator (LO). The mathematical description of an IQ mixer in the transmitter case can be formulated as follows.

Let the I component be a cosine signal with the frequency f_{IF}:

$$I = cos(2\pi f_{IF} t) = \frac{e^{j2\pi f_{IF} t} + e^{-j2\pi f_{IF} t}}{2} \tag{12}$$

And the Q component a sine signal with the frequency f_{IF}:

$$Q = sin(2\pi f_{IF} t) = \frac{e^{j2\pi f_{IF} t} - e^{-j2\pi f_{IF} t}}{2j} \tag{13}$$

The same goes for the LO, which will also have an in phase and a quadrature component with the frequency f_{LO}. LO_I should also be a cosine signal:

$$LO_I = cos(2\pi f_{IF} t) = \frac{e^{j2\pi f_{LO} t} + e^{-j2\pi f_{LO} t}}{2} \tag{14}$$

And LO_Q should be a sine signal:

$$LO_Q = sin(2\pi f_{IF} t) = \frac{e^{j2\pi f_{LO} t} - e^{-j2\pi f_{LO} t}}{2j} \tag{15}$$

According to figure 16 a), the two I components are multiplied and the two Q components are multiplied with each other. Afterwards, the sum out of this two

products is built.

The product of the I components will be:

$$RF_I = \frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} + e^{j2\pi(f_{IF}-f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}-f_{LO})t}) \quad (16)$$

And the product of the Q components will be:

$$RF_Q = -\frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} - e^{j2\pi(f_{IF}-f_{LO})t} - e^{-j2\pi(f_{IF}-f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t}) \quad (17)$$

In figure 16, the sum to be formed has a minus sign at the Q component. This minus sign is necessary if the upper side-band is to be selected. The following then applies to the upper side-band:

$$\begin{aligned} RF = &\frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} + e^{j2\pi(f_{IF}-f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}-f_{LO})t}) \\ &+\frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} - e^{j2\pi(f_{IF}-f_{LO})t} - e^{-j2\pi(f_{IF}-f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t}) \end{aligned} \quad (18)$$

Which leads to:

$$RF = \frac{1}{4}(e^{j2\pi(f_{IF}+f_{LO})t} + e^{j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t}) \quad (19)$$

And this can again be written in terms of cosine functions:

$$RF = \frac{1}{2}(e^{j2\pi(f_{IF}+f_{LO})t} + e^{-j2\pi(f_{IF}+f_{LO})t}) = cos(2\pi(f_{IF} + f_{LO})t) \quad (20)$$

When the minus sign at the Q component is changed to a plus, the lower side-band will be the output of the IQ mixer. The fact that the output frequency is limited to one side-band makes this type of modulation also called Single-Side-Band-Modulation.

On the receiver side (figure 16 b), both the I and Q components are obtained from the received signals. The signal is mixed once with the LO and once with the 90° shifted LO. At the end of each channel there is a low-pass filter, which then selects the lower band of the mixers output.
From a mathematical view, the mixer delivers the following signal for the I component. It is assumed that the incoming RF signal is a cosine signal.

$$I_{MIX} = \frac{e^{j2\pi f_{RF}t} + e^{-j2\pi f_{RF}t}}{2} \cdot \frac{e^{j2\pi f_{LO}t} + e^{-j2\pi f_{LO}t}}{2} \quad (21)$$

Which leads to the following cosine signals:

$$I_{MIX} = \frac{1}{2}(cos(2\pi(|f_{LO} + f_{RF}|)t) + cos(2\pi(|f_{LO} - f_{RF}|)t)) \tag{22}$$

With an appropriate low-pass filter, the base-band signal can be obtained, which would then be:

$$I = \frac{1}{2}(cos(2\pi(|f_{LO} - f_{RF}|)t)) \tag{23}$$

For the Q component, the incoming signal will be the same as for the I component, but the LO will be shifted by 90° and therefore the RF signal is mixed with a sinusoidal LO:

$$Q_{MIX} = \frac{e^{j2\pi f_{RF}t} + e^{-j2\pi f_{RF}t}}{2} \cdot \frac{e^{j2\pi f_{LO}t} - e^{-j2\pi f_{LO}t}}{2j} \tag{24}$$

This ca be further written as:

$$Q_{MIX} = \frac{1}{4j}(e^{j2\pi(|f_{LO}+f_{RF}|)t} + e^{j2\pi(|f_{LO}-f_{RF}|)t} - e^{-j2\pi(|f_{LO}+f_{RF}|)t} - e^{-j2\pi(|f_{LO}-f_{RF}|)t}) \tag{25}$$

Which leads to the following two sinusoidal components:

$$Q_{MIX} = \frac{j}{2}(sin(2\pi(|f_{LO} - f_{RF}|)t) + sin(2\pi(|f_{LO} + f_{RF}|)t)) \tag{26}$$

And again, using the low-pass filter the base-band Q component can be obtained as:

$$Q = \frac{1}{2}j(sin(2\pi(|f_{LO} - f_{RF}|)t))) \tag{27}$$

2.2 LimeSDR USB Board

2.2.1 Block Diagram

The block diagram of the LimeSDR is shown in figure 17.

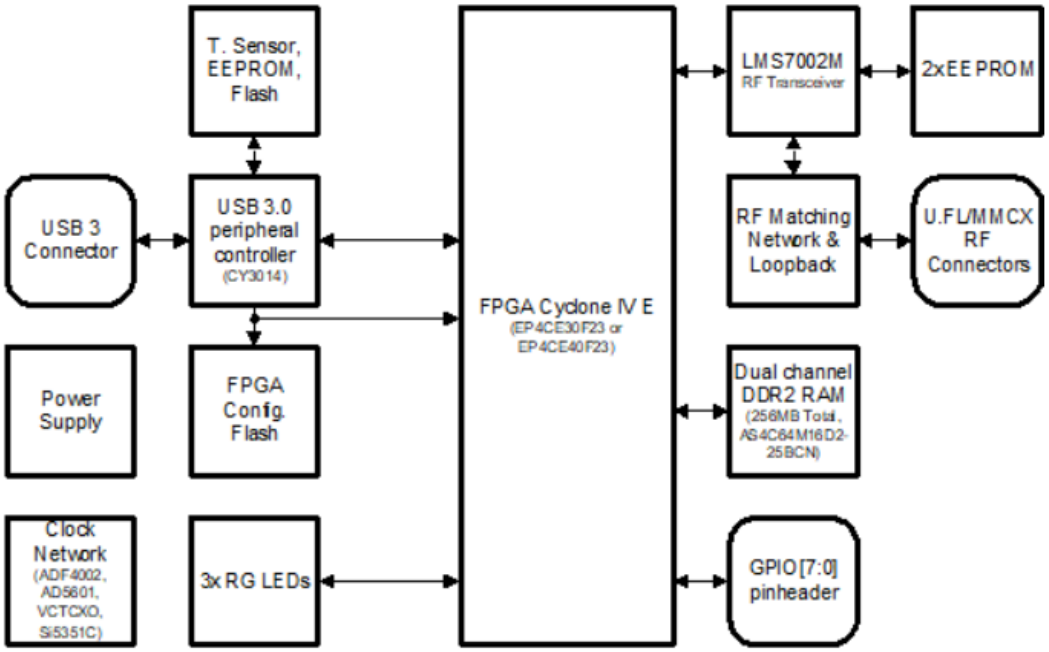

Figure 17: Block diagram of the LimeSDR board. Source: [10]

The heart of the SDR is the LMS7002 FPRF chip. In order to generate the desired signals and to control those from a host PC, a USB 3.0 controller is installed, which transfers the settings made on the host PC to the Altera Cyclon IV E FPGA. This in turn communicates the necessary settings to the LMS7002M FPRF chip.
RX data is then transferred from the LMS7002 FPRF via the FPGA and the USB 3.0 interface to the host pc and can be further processed.

Additionally the board is equipped with a 10 pin JTAG pin header for GPIO use. This is very useful for generating control signals for other peripherals. Please also refer to [10] for an overview of the different connectors on the board.

2.3 LMS7002M FPRF

The LMS7002M is a so called Field Programmable Radio Frequency (FPRF) multiple input multiple output (MIMO) transceiver IC, with a integrated microcontroller. It is equipped with two TX and RX channels and covers a broad frequency range from $100\,kHz$ to $3.8\,GHz$.

Figure 18 shows a block diagram of the LMS7002M transceiver IC.

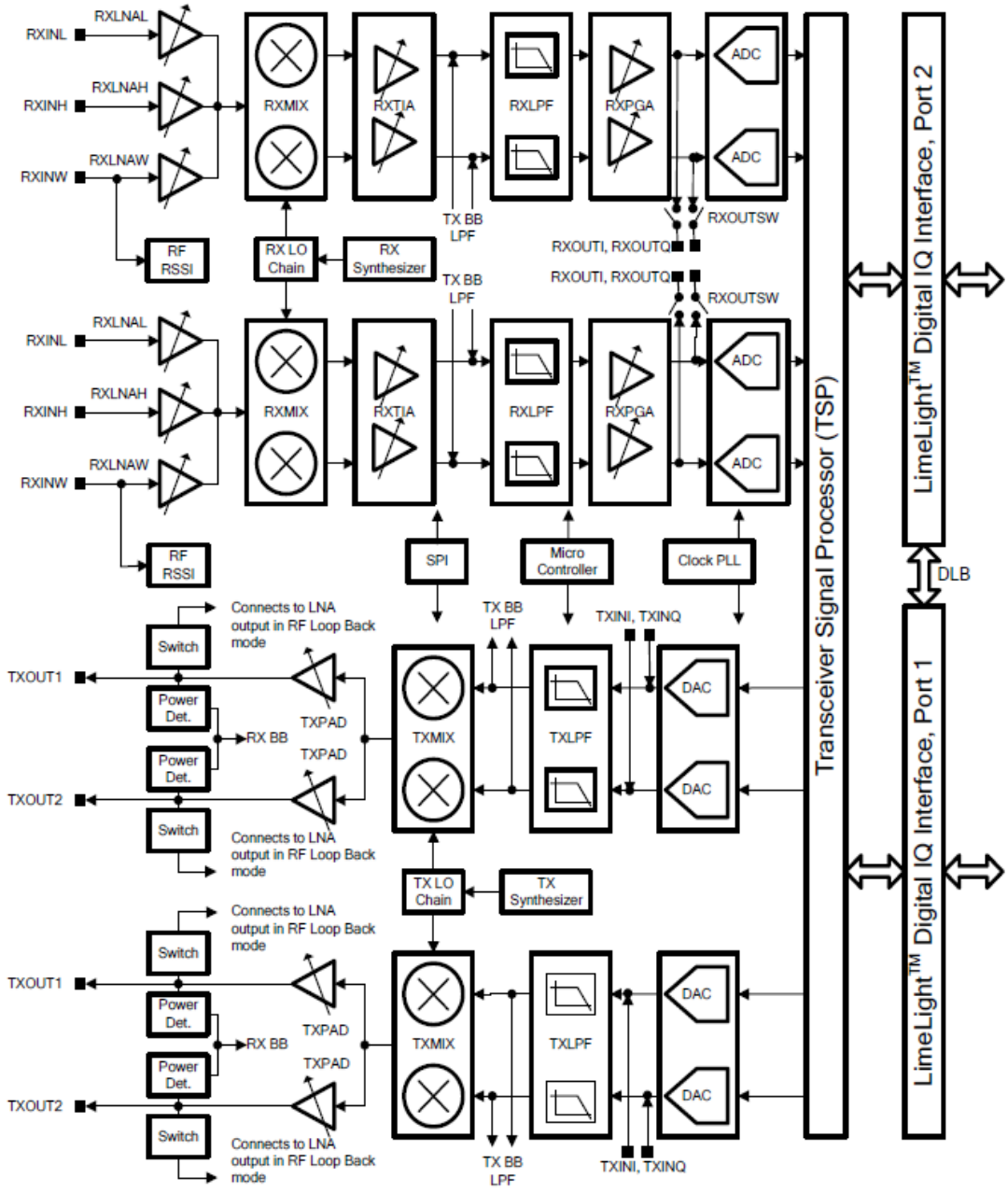

Figure 18: Block diagram of the LMS7002M. Source: [12]

In figure 18 the structure of the two RX and the two TX channels can be seen. Since this sketch contains a lot of information, the following blocks should provide a closer look at the module.

2.3.1 TX and RX path

The TX path starts in the LimeLight Digital IQ Interface, which provides the IQ data to the Transceiver Signal Processor (TSP). The TSP is divided into a TxTSP and a RxTSP (per channel). The TSP is basically responsible for ensuring that the I and Q channels are optimally prepared. Corrections to the IQ data of the LimeLight digital IQ interface can be made here. At best, the two tracks have exactly the same amplitude and do have a 90 degree phase-shift respectively.

After the digital IQ data has been optimally tuned, it is converted to two analogue tracks by a digital to analogue converter (DAC). A lowpass filter (TXLPF) can be used to limit the bandwidth upwards. The I and Q tracks are then mixed with a local oscillator to upconvert the signal from the base-band to the desired RF band. An amplifier (TXPAD) can then be used to tune the output power before it is output to a port accessible with UFL connectors.

To take a closer look into the TSP, figure 19 shows the building blocks of the TxTSP.

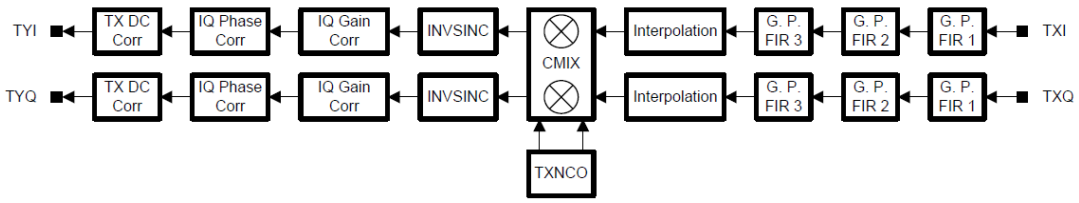

Figure 19: Block diagram of the LMS7002M's TxTSP. Source: [12]

The general purpose FIR filters (G. P, FIR 1-3) are not used by the pulse programmer software from [2]. According to the datasheet, the INVSINC can be used to compensate $\frac{sin(x)}{x}$ roll off, but was also disabled in this project. The mixer (CMIX) is also not used.
The blocks IQ Gain Corr, IQ Phase Corr and TX DC Corr are very important to calibrate the IQ data. With these, the generated signal can be optimised very well to a single frequency output. These can also be adjusted with the help of the software [2]. A description of the calibration procedure can be found in the appendix.

Finally, figure 20 shows the TX gain control and the TX mixer of the TX path. The TX mixer is responsible for mixing the base-band I/Q signals with the local oscillator (provided by the TXPLL) to generate the target RF frequency.

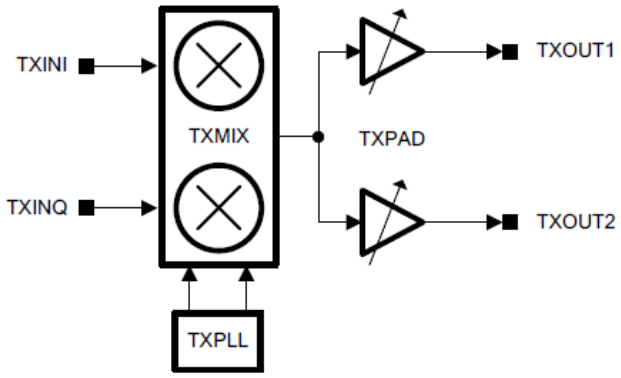

Figure 20: Block diagram of the LMS7002M's TX gain control including the TX mixer. Source: [12]

Now, a closer look at the RX path will be given. A received signal can be amplified immediately after it arrives at the port using the first amplifier (which are Low Noise Amplifiers (LNA)) of the RX path (RXLNA). Looking at figure 18 one can see that three inputs (RXINL, RXINH, RXINW) are connected to the same mixer via three separate LNAs. The reason for this is that the different inputs have different input matching-networks, which are tuned to different frequency ranges. Since only frequencies below 200 MHz were used in this work, only the RXINL (L for low) is used.

Afterwards, the received signal is mixed down to the base-band with the same local oscillator that is used to generate the transmitted signal. The signals are divided into an I and a Q component as explained in 2.1.3. After the down-conversion, there is another amplifier (RXTIA). In the base-band, the band can still be limited with the help of a low-pass filter (RXLPF). This is followed by the last of the three amplifiers in the RX path (RXPGA) before the signals are then digitised with an analogue to digital converter (ADC). The signals, which are now digital, are then passed on to the RxTSP.

Figure 21 shows the block diagram of the RxTSP.

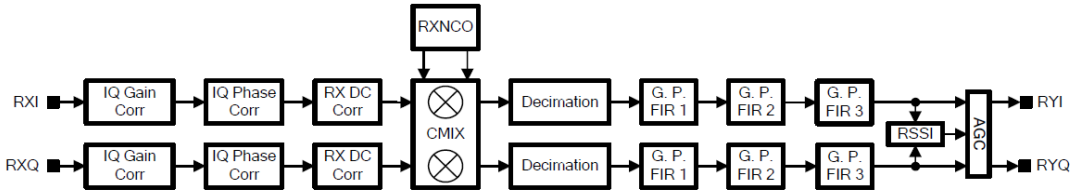

Figure 21: Block diagram of the LMS7002M's RxTSP. Source: [12]

As with the TX path, the general purpose FIR filters (G. P, FIR 1-3), and the CMIX is not used. The blocks IQ Gain Corr, IQ Phase Corr and TX DC Corr in the RX path can also be adjusted with the software from [2]. However, these settings were left at the default settings, as there were no problems with the reception of the signals.

The RX gain is always divided between the three RX amplifiers (RXLNA, RXTIA, RXPGA) mentioned above. Figure 22 shows the block diagram of the RX gain control and the RX mixer. The RX mixer is used to convert the received RF signal into a I and a Q base-band component using the LO provided by the RXPLL.

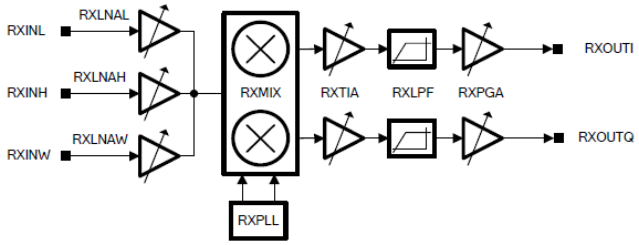

Figure 22: Block diagram of the LMS7002M's RX gain control and the RX mixer. Source: [12]

2.3.2 LimeSDR RF Performance

Figure 23 shows the LimeSDRs RX performance for the $RX1\_L$ port. The table describes different hacks to perform a different RX performance. The components mentioned here are part of the matching network for the $RX1\_L$ port (see also appendix figure 137 to see where they are located). The unit $dBFS$ describes the input power in dBm normalized to the full scale (the maximum).

	MN19	MN17	MN20	MN18	C14
Default (900MHz)	1.2pF	510pF	510pF	8.2nH	470pF
EasyFix1	1.2pF	510pF	510pF	Remove	470pF
EasyFix2	Remove	100nF	100nF	Remove	100nF
Broadband (100kH-400Mhz)	1pF//82R	8.2nH	8.2nH	1pF	100nF

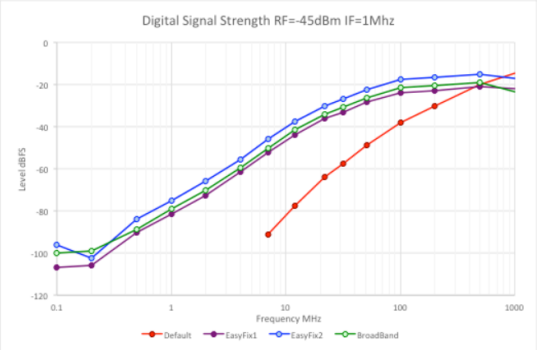

Figure 23: LimeSDR RX RF performance for the $RX1\_L$ and how to improve it. Source: [22].

Figure 24 shows the LimeSDRs TX performance. $TX1\_1$ and $TX1\_2$ are the two TX ports of TX channel 1 of the LimeSDR (there is a second TX channel with also two output ports).

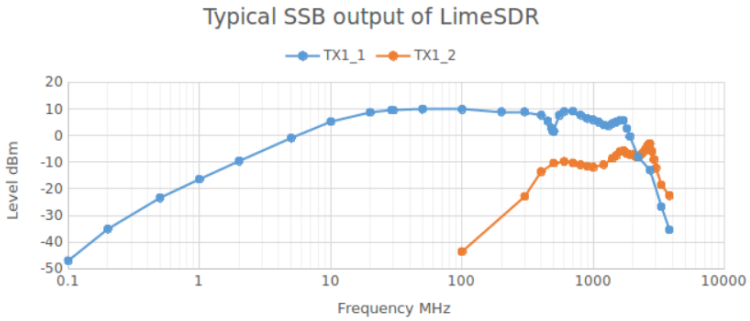

Figure 24: LimeSDR TX RF performance for SSB outputs. Source: [22].

2.4 Open Source Files: C++ Routine and Python script

As depicted in figure 13, the open source files from [2] consist out of a C++ routine which is responsible for setting all registers on the chips and to stream and save data. A python module used to communicate with the C++ code is

also delivered, through which a pulse experiment can be set up with a python script. An introduction how to set up the working environment can be found in the appendix.
The general idea of the project is to perform a pulse sequence at a certain frequency. The selected IF frequency is mixed with the LO frequency in the TX path and the sum of the LO and IF frequencies forms the transmitted RF signal. On the RX side, the RF signal is then received, mixed down and then digitised. Figure 25 shows the IF spectrum after down mixing and sampling (single peak due to complex sampling). The LO frequency can be added to the frequency axis to show up the RF frequency which was received before down mixing. The C++ routine saves the time domain data to a hdf5 file where the data points are summed up before they are saved (with the selected number of averages). Many SDR settings such as TX and RX gain, receive bandwidth, local oscillator and base-band frequency, etc. can be set via a python script. Furthermore, it is possible to generate any number of pulses with any averaging number and repetition rate with minimum a time resolution of $33\,ns$ (depending on the sample rate, here it was always set to $30.72\,MHz$).
The output pulse pattern can also be accompanied by a GPIO signal, which can be used as a gate (e.g. for the power amplifier (PA)). In total there are four pins which can provide a GPIO signal, in this work only one of them was used, but it is possible to use all four. However, in order to switch the GPIO signals with the pulse sequence, a small change must be made to the default FPGA code. An image which can be transferred to the SDR with the help of 'LimeSuiteGUI' is also included in [3] (name: $LimeSDR-USB\_lms7\_trx\_HW\_1.4.rbf$). For a detailed instructions: see appendix.

In order to use the C++ routine, it has to be compiled first. This has to be done only once or whenever you have made a change to the code. Therefore, g++ is needed on the Ubuntu system. To compile the C++ file, one line from the header of the C++ file has to be executed in the command line of the terminal (see also appendix, figure 117).

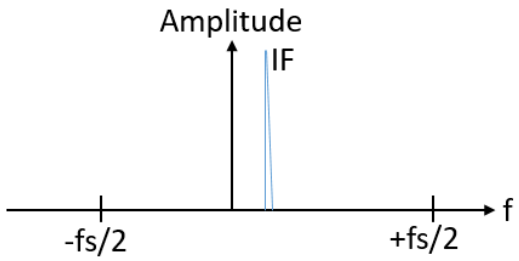

Figure 25: IF Spectrum illustration.

2.4.1 Python Command Table

The following table should give an overview of the parameters which are commonly used in the python sequence scripts. Only those used in this work are listed here. Some commands were not needed but can be found in the C++ code (there is also a short comment for each command).

Table 1 shows the Python Command table.

Table 1: Python Command Table with descriptions.

Python Commands and descriptions	
Command	Description / Comment
sra	IF Sampling Rate in Hz
lof	Local Oscillator (LO) frequency in Hz
rlp	RX lowpass filter bandwidth in Hz
tlp	TX lowpass filter bandwidth in Hz
rgn	RX Gain in dB
tgn	TX Gain in dB
tdq	TX DC correction of Q channel
tdi	TX DC correction of I channel
tgi	TX Gain Correction I
tgq	TX Gain Correction Q
tpc	TX phase correction (I/Q)
rgi	RX Gain Correction I
rgq	RX Gain Correction Q
rpc	RX phase correction (I/Q)
npu	Number of Pulses
pdr	Pulse duration in s
pof	Pulse Offset in Samples (Sa)
pam	IF Pulse Amplitude
pfr	IF Pulse Frequency in Hz
pph	IF Pulse Phase
pcn	Number of Phase-Cycles
pcl	Level of Phase-Cycle
pba	Phase Cycles before Averaging (if >0)
t3d	Trigger 3 Timing (GPIO 3)
nrp	Number of Repetitions
nav	Number of Averages (per repetition)
trp	Repetition time
tac	Acquisition Time in s
fpa	Filename Pattern
spt	Save Path
noi	Don't initialize if > 0
nos	Don't save if > 0

2.4.2 Python Script Example

The following python snapshots in this section and in section 'Signal Processing' together result in a pulse sequence, i.e. the timing for the sequence of pulses and the subsequent necessary blanking and selection of the acquisition time and acquisition window.

Figure 26 shows the timing diagram for a sequence with two pulses, which is realized by the pulse script and the post processing script. Figure 27 shows a snapshot of a python script which generates two pulses. Note: figure 27 and figure 28 are one python file, it is just split due to readability and to show that there is a pulse generating part and a post processing part.

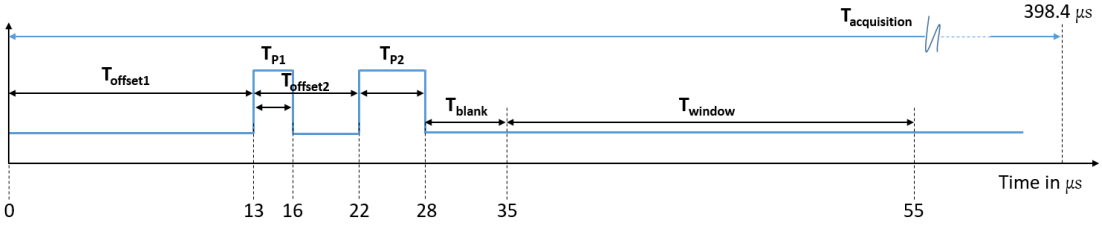

Figure 26: Timing diagram of a sequence with two pulses. The repetition time T_R was set to $T_R = 1\,ms$.

```
import numpy as np
import limr
from scipy.fftpack import fft, fftshift, fftfreq

l = limr.limr('./pulseN_test_USB.cpp');
tgtfreq = 83.56e6
if_frq = 1.2e6

l.lof = tgtfreq-if_frq                          # LO frequency
l.sra = 30.72e6                                 # Sampling Rate
l.nav = 1000                                    # number of averages

l.nrp = 1                                       # number of repetitions

l.trp = 1.0e-3                                  # repetition time
l.tac = 82e-6                                   # acquisition time


l.t3d = [1, 10, 53, 10]                         # GPIO Trigger 3 [Enable, pad0, shift, pad2]


l.nos = 0                                       # do not save if >0

l.pfr = [if_frq, if_frq]                        # pulse frequency
l.pdr = [3e-6, 6e-6]                            # pulse  duration: T_Pulse1 and T_Pulse2
l.pam = [1.0 , 1.0]                             # relative pulse amplitude


l.pof = [300, np.ceil(9e-6*l.sra)]              # pulse offset: T_offset1 and T_offset2
l.npu = len(l.pfr)                              # number of pulses


l.rgn = 55.0                                    # RX gain
l.tgn = 40.0                                    # TX gain

l.rlp = 3.0e6                                   # RX lowpass BW
l.tlp = 130.0e6                                 # TX lowpass BW

l.spt =  './SpinEcho/data'                      # directory to save to
l.fpa = 'measurement'                           # filename pattern
```

Figure 27: Snapshot of a python script producing two pulses. The structure and content is based on the sequence presented in [2].

The first step of a pulse-script using python is to import the class 'limr' which can be found in [3]. This class is responsible for parsing all of the commands (see table 1) to the C++ routine which is also available in [3].

The sampling rate was always set to $30.72\,MHz$ in this project, as with this choice a stable communication between host PC and LimeSDR was achieved and the measured signals have a sufficiently high resolution with respect to the number of recorded data points. In principle, the sample rate can be divided down as desired but this was not tested in this work. The upper limit is $61.44\,MHz$, which is limited by the USB 3.0 interface. However, it is not recommended to use this sample rate, since packet loss already occurs between host PC and SDR. The maximum (reliably) usable sample rate is therefore $30.72\,MHz$.

The intermediate frequency (IF) and the local oscillator frequency (LO) can be set separately. Furthermore, the number of averages and the number of repetitions of the experiment can be set. The acquisition time must be set in a way that it is smaller than the repetition time. The IF frequency in this example was set to $1.2\,MHz$ since this choice allows a narrowband RX low-pass filter to be selected ($3\,MHz$) and thus no higher frequency interference can disturb the spectrum. This filter limits the RX bandwidth and thus also the noise. It is recommended to not set the IF frequency lower than $1\,MHz$, as it will then get closer to the LO frequency and the distinction between LO leakage and IF signal may become more difficult. No specific studies have been performed for this, but $1.2\,MHz$ has been shown by experience to be a good choice. The LO frequency was set to $82.36\,MHz$ (because this is the difference to the target frequency of $83.56\,MHz$), the repetition time to $5\,ms$ and the experiment is averaged 1000 times.
The necessary data stream is packed into buffers by the C++ routine, which always have a size of 4080 samples. By default, this buffersize is multiplied by 3 ($3 \cdot 4080 = 12240$ samples). A standard buffer of 4080 samples then leads to an acquisition time of:

$$T_{acq} = \frac{4080}{30.72 \cdot 10^6} = 132.8\,\mu s \tag{28}$$

Thus, for $l.tac \leq 132.8\,\mu s$, the minimum acquisition time of $132.8\,\mu s$ would be used, which is the case here for $l.tac = 82\,\mu s$. As already mentioned, this is then multiplied by three and gives a total acquisition time of:

$$T_{acq} = \frac{4080}{30.72 \cdot 10^6} \cdot 3 = 398.4\,\mu s \tag{29}$$

So the $82\,\mu s$ are just a synonym for 'set the smallest possible acquisition time'. The LimeSDR records from the start of the experiment until the selected acqui-

sition time and sums up the data points in time domain with the given number of averages before saving it into a hdf5 file. In this way, the size of the stored files can be kept as small as reasonably possible according to the sequence parameters.

The GPIO output used (l.t3d) is output simultaneously with the pulse. However, it is possible to shift it or to lengthen or shorten it. The first vector entry means 'enable' (so set to 1 if enabled, 0 if disabled), the second one is the 'padding0' and means padding before the rising edge (can be negative and must be if the GPIO should be opened before the pulse). The third entry is the 'shift' which causes a shift between the GPIO signal and the pulse and the fourth entry is 'padding2' (can be negative) which can lengthen or shorten the time to onset of the falling edge of the GPIO signal.

The script is structured in such a way that in the case of several pulses, the nth entry in the vectors specifying the pulses is always assigned to the respective pulse. In the code excerpt above, the arrays that define the pulses are l.pfr, l.pdr, l.pam and l.pof. Looking at these vectors, this means that the first and second pulse have the same frequency, that the first has a duration of $3\,\mu s$ and the second a duration of $6\,\mu s$, that both have the maximum amplitude (namely 1.0) and that the first pulse has an offset of 300 sample points and the second is shifted by $9\,\mu s$ with respect to the starting point of the first pulse.

It is also possible to set a TX low-pass filter and an RX low-pass filter. The selectable bandwidth is the $-3\,dB$ bandwidth. The TX and RX signals can be amplified by on-board amplifiers. The amplification already has the unit dB for both, the RX and TX gain. Some of the TX gains were tried out and the respective output powers were calculated in table 2.

2.4.3 Signal Processing

The presented code in figure 27 produces the necessary excitation pulses and stores summed up time domain data acquired at the RX port. A practicable sequence also includes the correct evaluation of the measured data, which will be described here.

Figure 28 shows a standard post processing script for a Spin-Echo Sequence.

```
#reads back the file which was recently saved
l.readHDF()

#evaluation range, defines: blanking time and window length
evran = [35, 55]

#pick values from the selected range out of the time domain data from hdf5 file
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

#time domain x and y data
tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

#correcting a offset in the time domain by subtracting the mean
tdy_mean = tdy-np.mean(tdy)

#fft of the corrected time domain data
fdyl = fftshift(fft(tdy_mean,axis=0),axes=0)

#fft freq and fft shift is here used to scale the x axis (frequency axis)
fdxl = fftfreq(len(fdyl))*l.sra/1e6
fdxl = fftshift(fdxl)

#scaling factor which converts the y axis from points into uV

plt.figure(1);
plt.plot(tdx,tdy_mean)
plt.xlabel("t in µs")
plt.ylabel("Amplitude in µV")
plt.show()

#get LO frequency and add it to the base band fft x-Axis in order to illustrate the applied frequency
#for single side spectrum and shift (only single frequency)
lof=l.HDF.attr_by_key('lof')

for i in range(0, len(fdxl)):
    fdxl[i] = fdxl[i]+lof[0]/1e6

#this passage can shift the data to be plotted, e.g. to not show higher frequency components
#which where not measured
shifter = 12
stopper = 270

#here the right side of the spectrum is selected
#FFT is always done from -fs/2 to +fs/2; left side of spectrum will not show signal peaks
y=abs((fdyl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]))
x=fdxl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]

plt.figure(4);
plt.plot(x, y)
plt.xlabel("f in MHz")
plt.ylabel("Amplitude in µV")
plt.show()

#print std (for SNR determination -> noise analysis without sample)
print("std rms frequency domain next to peak X: " + str(np.std(y)))
#print max of fft (for SNR evaluation - should give peak maximum)
print("MAX of Signal: " + str(max(y)))
```

Figure 28: Snapshot of a python based post-processing script.

As already mentioned above, both the pulse generating part and the post processing part are required to realize a sequence. The total acquisition time for the experiment can be read from the equation above which is $398.4\mu s$. In order to set the blanking and the window length, the data has to be read back from the hdf5 file and then the evaluation window has to be set.
The time domain data is loaded back into the python script and an DC offset correction is done by subtracting the mean value of the amplitude of the time domain data. After that, the Fast Fourier Transformation (FFT) is applied to the data and then it is shifted (using fftshift()) in a way that the zero frequency component is in the center of the spectrum. The frequency axis is extracted by using the fftfreq() function and multiplying the vector by the sample rate. To get a scale that is symmetrical about $0\,Hz$, the frequency axis must also be shifted with fftshift(). The Local Oscillator (LO) frequency is then added to each frequency point to represent the actual transmitted frequency. In addition, the

left side of the spectrum is usually not plotted, as it does not contain any signal information. Typically, the interesting part with about $\pm 500\,kHz$ around the excitation frequency is cut out and only this section is plotted. For possible SNR evaluations the standard deviation and the maximum of the excised frequency range are printed to the console. The maximum is important for the evaluation of the sample's signal strength. The standard deviation is used for the evaluation of the noise, wherefore the sample is removed.

It should be highlighted, that the received signal is always mixed down before it is digitised (see IQ Mixer) and the LO frequency is added by post processing. The sampling rate then defines the limits of the spectrum calculated with the help of the FFT (see figure 25). Here a spectrum of $\pm\frac{f_s}{2}$ is always calculated and then the part of interest is cut out in post-processing (figure 28) and the LO frequency is added to these IF values.

Figure 29 should illustrate the main signal processing steps.

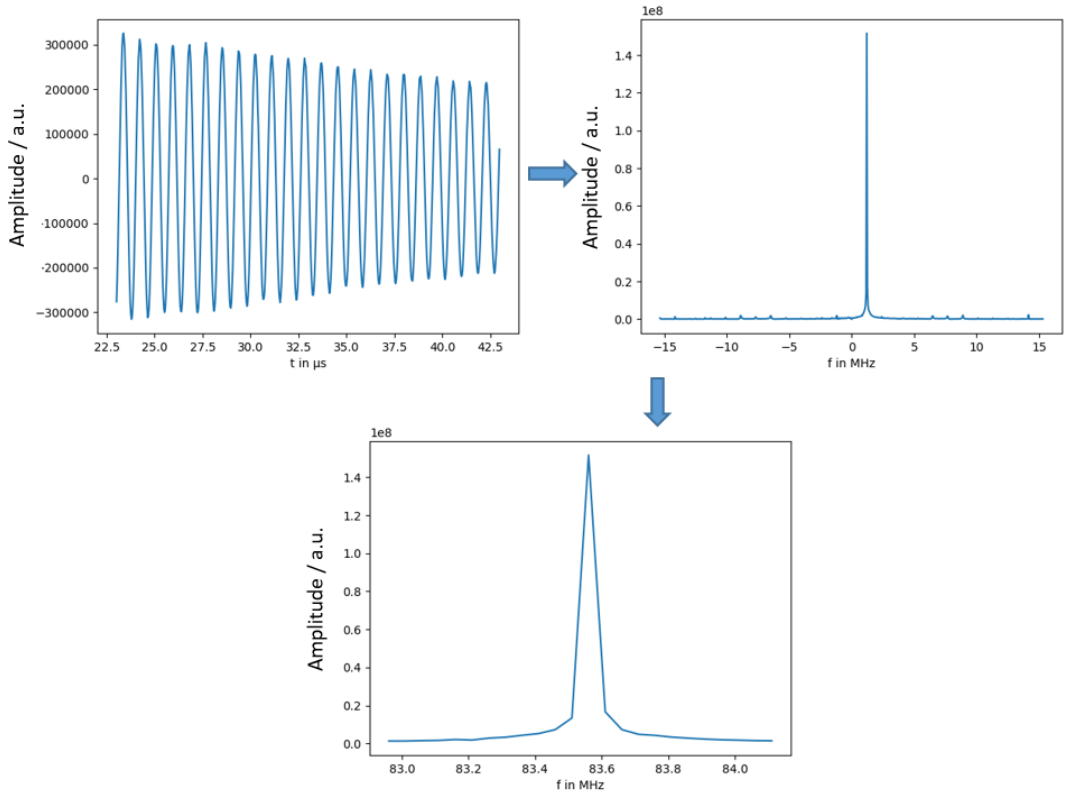

Figure 29: Typical signal processing steps.

2.5 Preparation for using the LimeSDR's GPIO

The GPIO signals already mentioned must be tapped from the LimeSDR board. Furthermore, the GPIO signals have an amplitude of 3.3 V and must be shifted to about 5 V in order to supply the peripheral devices such as the power amplifier with a gate signal. On the one hand, the power amplifier used requires these signal amplitudes, on the other hand, a driver circuit should tap the signals from the board so that the SDR board does not have to drive many external devices at the GPIO port (limited power).
In this work, only one of the four available GPIO pins was used, which is marked in figure 30 on the LimeSDR board. There is no special reason for using the pin labelled with 'GPIO 3' - any other pin could be used in a similar manner.

Figure 30: LimeSDR board with the GPIO Pin used marked in red.

In order to use the GPIO signal as a trigger or as a gate, two wires need to be connected to the two marked pins in figure 30. Best is to use a 1.27 mm pitch PCB socket and to solder the two cables onto it.

2.5.1 GPIO Level Shifter PCB

To shift the 3.3 V GPIO signal to 5 V, a fast rail to rail comparator (TLV3501AID, Texas Instruments) was used. The TLV3501AID has a very short propagation delay of only 4.5 ns and the rise and fall time is 1.5 ns according to the datasheet. Further it was selected as it is a rail-to-rail comparator, which means it can output voltages up to the selected supply voltage (no voltage drop). Therefore, a PCB was designed using 'EAGLE'. Figure 31 shows the schematic of the GPIO level shifter.

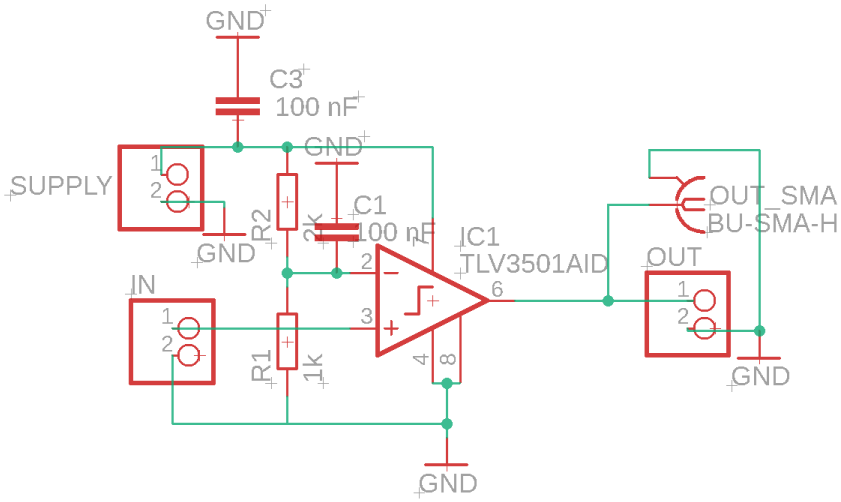

Figure 31: Schematic of the GPIO level shifter PCB.

Figure 32 shows the boardplan of the GPIO level shifter.

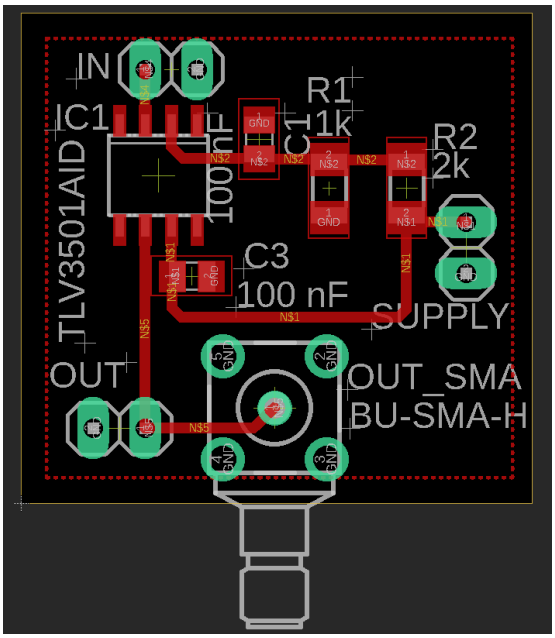

Figure 32: Boardplan of the GPIO level shifter PCB.

The functional principle of this circuit is to quickly raise the gate signal to a level of $5\,V$. The comparator is dimensioned in such a way that from an applied level of approx. $1.67\,V$ (given by the voltage divider of $R_1 = 1\,k\Omega$ and $R_2 = 2\,k\Omega$ at the positive comparator input) it outputs the $+5\,V$ supply voltage at the output.

2.6 Functionality check of the LimeSDR as Transceiver

To check the functionality of the software from [2] and the previous installations and the whole setup, the $TX1\_1$ channel of the SDR can be connected to an oscilloscope. It is also advisable to examine the signal in the frequency domain using a spectrum analyser in order to assess the need for calibration and, if necessary, to make corrections immediately.

2.6.1 Time Domain Measurements, Oscilloscope

Figure 33 shows the measurement setup for the transmit check in time domain. It is advisable to connect the GPIO output, which is the gate signal, to the second oscilloscope channel and to set the trigger onto it.

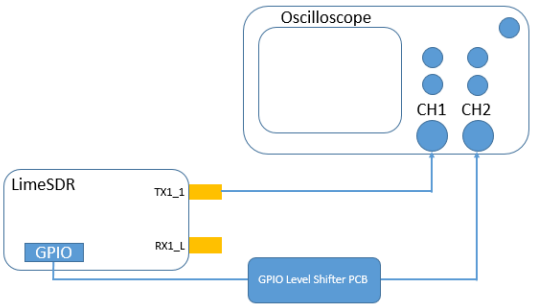

Figure 33: Measurement setup to check the transmit channel of the LimeSDR in time domain.

2.6.2 Frequency Domain Measurements, Spectrum Analyzer

Figure 34 shows the measurement setup for the transmit check in frequency domain. The CW test signal used was output using 'LimeSuiteGUI'. Remark: see appendix for the TX calibration guide where it is also explained how a CW test signal can be produced using the 'LimeSuiteGUI'.
A pulsed FID sequence was also tested, with the TX gain of the SDR increased to about $50\,dB$, the pulse length set to $600\,\mu s$ and the repetition time to $1\,ms$. This was needed because otherwise no signal was visible on the spectrum analyzer due to the pulsed operation and the attenuator.

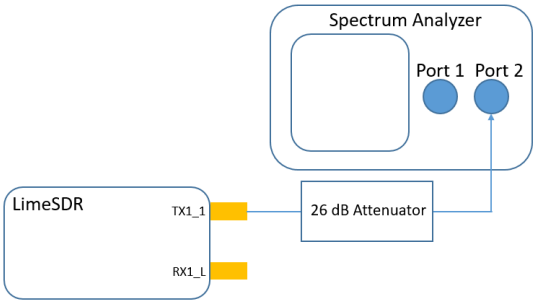

Figure 34: Measurement setup to check the transmit channel of the LimeSDR in frequency domain. A 26 dB attenuator was used for safety reasons.

2.6.3 Loopback to check the LimeSDRs RX path

Figure 35 shows the measurement setup to check the LimeSDR's RX path with the pulses generated in the TX path. Remark: this loopback was performed with the later explained pulse sequences FID, Spin-Echo, Spin-Echo with phase-cycling and Composite Pulse (figures 52, 53 and 55). Results can be seen in the figures 70 to 73.

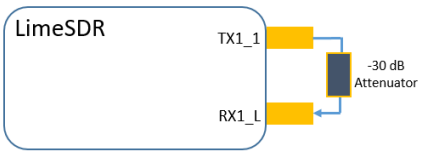

Figure 35: Setup to check the RX path of the LimeSDR with the generated pulses from the TX path.

2.6.4 Output voltage levels of the SDRs TX path

In the course of the transmit checks, the output voltages at different TX-Gain values (between 40 dB and 80 dB) were measured with an oscilloscope and converted to dBm. Table 2 shows the table with the different TX values.

Table 2: TX Gain of the LimeSDR.

TX Gain of the LimeSDR			
Python Value in [dB]	V_{peak} in mV	V_{rms} in mV	V in dBm
30	13	9.19	−27.72
40	40	28.28	−17.96
50	130	91.92	−7.72
60	400	282.84	2.04
70	1300	919.24	12.28
80	1500	1060.66	13.52

2.7 Tuning and Matching

2.7.1 Theory

Tuning and Matching refers to the impedance matching of a network to a standardized impedance in order to achieve maximum transmission and minimum reflection of an RF-signal from a source to a load. In NQR and NMR experiments, it is necessary to match the used coil to the measuring device depending on the applied frequency. The standardized impedance in RF-Electronics is usually $50\,\Omega$ (there are also systems that have $75\,\Omega$ termination impedance, but here a $50\,\Omega$ system is present).

The difference between NQR and NMR with respect to the applied frequencies is that NMR systems often do not need to make large changes to the transmission frequency, as NMR Spectroscopy targets hydrogen-containing compounds and this has a fixed Larmor frequency in the presence of a constant external magnetic field ($\gamma = 42.577 \frac{MHz}{T}$).
For NQR nuclei, however, the spin transitions (of one nucleus if it shows several transitions and when investigating different compounds containing quadrupole nuclei anyway) can be very far apart (tens of MHz), so the coil used must be adjustable over a very wide frequency range.

The matching-network typically used consists of a tuning capacitor (C_T) connected in parallel to the coil and a matching capacitor (C_M) connected in series to the parallel resonator. The tuning capacitor has the task to transform the impedance of the coil to $50\,\Omega$ (real) and the matching capacitor is used to eliminate the imaginary part.

Figure 36 shows a Tuning- and Matching-Network which consists out of two capacitors and the sample coil.

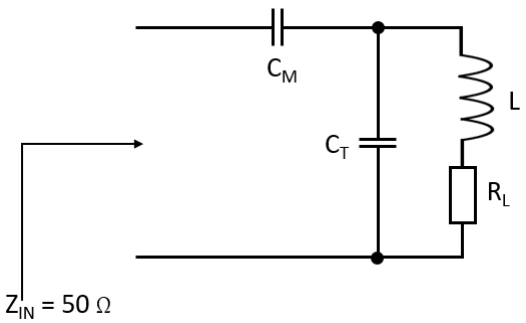

Figure 36: Schematic of a Tuning- and Matching-Network with two capacitors and the sample coil.

The implementation of the capacitors can be realized in different ways. On the one hand, tube trimmer capacitors can be used, which can be adjusted either manually or automatically with stepper motors. Both of these options require a mechanical tuning process.
However, there is also the possibility of building electronically tunable networks by replacing the capacitors with varactor diodes. These change their junction capacitance depending on the voltage applied.

In this work, a manually tunable coil was primarily used. However, the foundations were also laid for the use of electronically tunable probe heads (already available at the Institute of Biomedical Imaging from previous research work).

2.7.2 Example

The circuit shown in 36 is a frequently found implementation of a tuning and matching circuit, since the two capacitors can be realised as variable components. In principle, there are many different possibilities for a matching network. However, one must pay attention to the practicability of this particular form, as it is necessary to be able to adjust the coil over a large frequency range, as already mentioned. There are also different ways to calculate a matching network. The method now presented shows how it can be calculated via the quality factor.

Figure 37 shows the illustration of the matching problem. Basically, the TX path can be seen as a source with an internal resistance of $R_S = 50\,\Omega$.

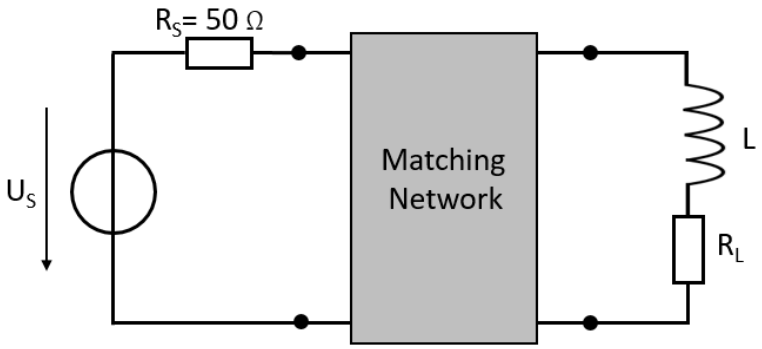

Figure 37: Illustration of the matching problem.

The idea now is to transform the real part of the coil (R_L) to the 50Ω of the source using a network. The form realised here is a so called T-Matching-Network, where the coil L acts as part of the network. For the calculation, a coil with $L = 300\,nH$ and $R_L = 3\,\Omega$ at a frequency of $f = 83.5\,MHz$ is assumed.

Figure 38 shows the illustration of the different components of the T-Matching Network. X_{S1} and X_{S2} denotes reactive resistances and B_P denotes reactive conductance. As mentioned above, the coil becomes part of the matching network, and this can be done in two ways. Either by compensating the reactance caused by the coil or by removing the reactance by resonance which would require a capacitor. Here, the compensation variant is applied, thus the coil itself will be X_{S2}. The solving process was taken from [28].

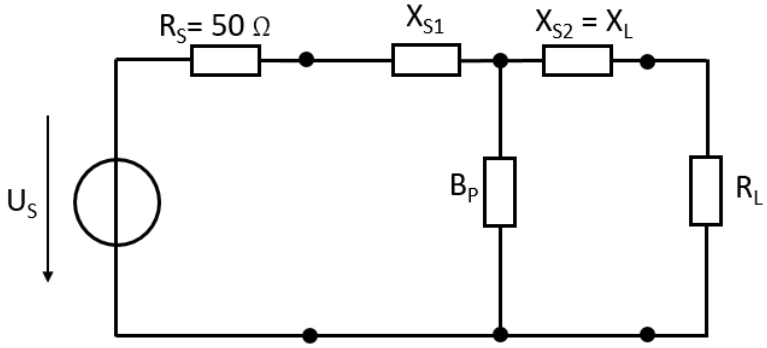

Figure 38: Illustration of the T-Matching Network.

Conceptually, the network is split in the middle of B_P and both sides are matched to a virtual resistor R_V. This is shown in 39. This takes advantage of the fact, that in a matched network, matching also prevails in any cut through the network.

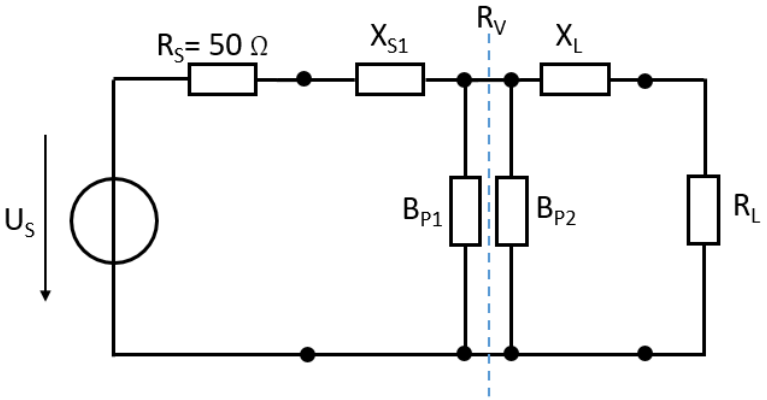

Figure 39: Illustration of the T-Matching Network.

The first step is to calculate the total quality factor given by the resistors R_S and R_L.

$$Q_{total} = \sqrt{\frac{R_S}{R_L} - 1} = \sqrt{\frac{50}{3} - 1} = \pm 3.96 \tag{30}$$

And the quality factor of the coil L:

$$Q_L = \frac{X_L}{R_L} = \frac{\omega L}{R_L} = \frac{2\pi \cdot 83.5 \cdot 10^6 \cdot 300 \cdot 10^{-9}}{3} = 52.46 \tag{31}$$

The virtual resistor R_V is calculated as:

$$R_V = (1 + Q_L^2) \cdot R_L = (1 + 52.46^2) \cdot 3 = 8259.15\,\Omega \tag{32}$$

Then, Q_1 for the left side of the network is calculated:

$$Q_1 = \pm\sqrt{\frac{R_V}{R_1} - 1} = \pm\sqrt{\frac{8259.15}{50} - 1} = \pm 12.81 \tag{33}$$

There are now two possibilities for B_P and X_{S1}. The calculation of the two solutions for B_P is as follows:

$$B_{P,solution1} = \frac{|Q_1| + Q_L}{R_V} = \frac{12.81 + 52.46}{8259.15} = 7.903\,mS \tag{34}$$

$$B_{P,solution2} = \frac{-|Q_1| + Q_L}{R_V} = \frac{-12.81 + 52.46}{8259.15} = 4.801\,mS \tag{35}$$

The calculation of the two solutions for X_{S1} is as follows:

$$X_{S1,solution1} = |Q_1| \cdot R_1 = 12.81 \cdot 50 = 640.5\,\Omega \tag{36}$$

$$X_{S1,solution2} = -|Q_1| \cdot R_1 = -12.81 \cdot 50 = -640.5\,\Omega \tag{37}$$

And then, the L and C elements can be calculated from the reactances and conducatances:

$$C_{P,solution1} = \frac{B_{P,solution1}}{\omega} = \frac{4.801 \cdot 10^{-3}}{2\pi \cdot 83.5 \cdot 10^6} = 15.06\,pF \tag{38}$$

$$C_{P,solution2} = \frac{B_{P,solution2}}{\omega} = \frac{7.903 \cdot 10^{-3}}{2\pi \cdot 83.5 \cdot 10^6} = 9.15\,pF \tag{39}$$

$$L_{S1,solution1} = \frac{X_{S1,solution1}}{\omega} = \frac{640.5}{2\pi \cdot 83.5 \cdot 10^6} = 1.22\,\mu H \tag{40}$$

$$C_{S1,solution2} = \frac{1}{X_{S1,solution1} \cdot \omega} = \frac{1}{640.5 \cdot 2\pi \cdot 83.5 \cdot 10^6} = 2.98\,pF \qquad (41)$$

Since the solution with two capacitors is more practical and this solution was targeted, the matching network for $83.5\,MHz$ with two capacitors is shown in figure 40. The solutions with the same index always belong together, which means that solution 2 was used. Solution 1 would require a coil for X_{S1}.

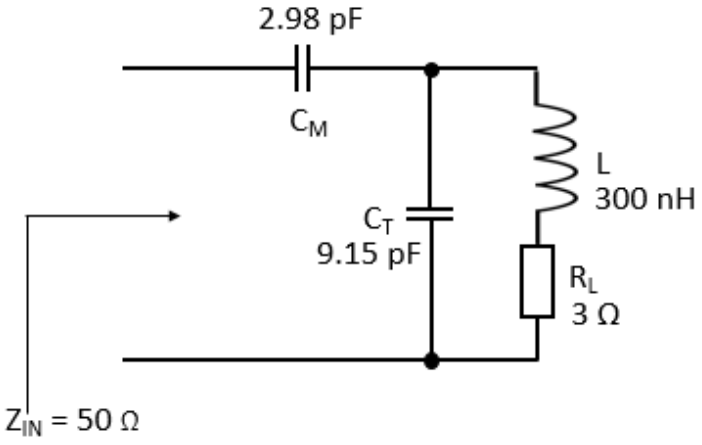

Figure 40: Illustration of the solution of the matching task.

2.8 S-Parameters

S-parameters are often measured to characterise RF systems. In this work, S-parameter plots are determined for the different stages for the characterization of the systems assemblies. This section is intended to give an overview of these. Basically, S-parameters of an electrical multi-port are used to describe the system's properties. An electrical multi-port always has a number of $ports^2$ S-parameters, which are usually combined in a matrix. Often, electronic circuits have two ports (input, output) and thus have 4 S-parameters.
The S_{11} parameter describes the input reflection of the system. This means, a signal is generated by a Network-Analyzer (NWA) and fed into the input of the system. Then, the NWA measures back how much of the signal is reflected. This parameter describes how well the input of the system is matched to the standardized impedance (often $50\,\Omega$). Analogous to this is the S_{22} parameter which is measured identically, except that the output of the system is examined.

The S_{21} parameter is analogous to the gain of the system. A signal is fed from the NWA to the input of the system and measured at the output of the system. With the help of the signal at the output, the NWA determines the gain. Again, the S_{12} parameter is determined in exactly the opposite way. Here, the signal is fed in at the output of the system and measured at the input of the system. This parameter therefore represents the reverse gain. The following figures show the measurement setups for the different S-parameters. The reverse gain was not determined, as only the input and output matching and the gain was of interest. The reverse gain would for example be necessary if stability analysis would be of interest.

Figure 41 shows the measurement setup for a S_{11} measurement.

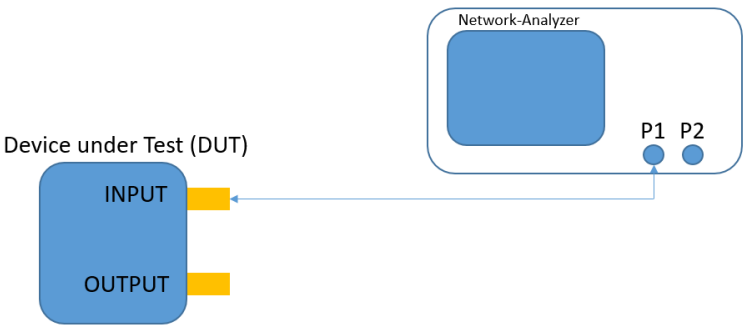

Figure 41: Schematic of a S_{11} measurement.

Figure 42 shows the measurement setup for a S_{22} measurement.

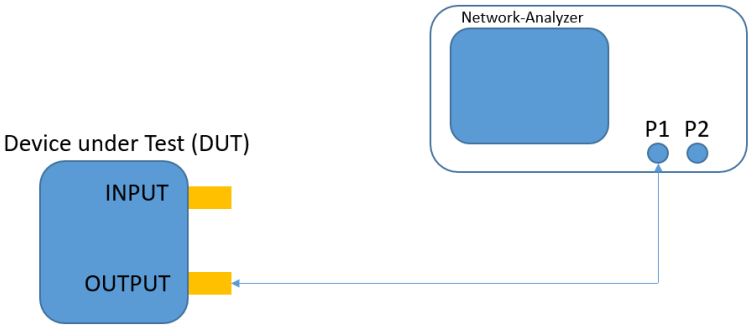

Figure 42: Schematic of a S_{22} measurement.

Figure 43 shows the measurement setup for a S_{21} measurement.

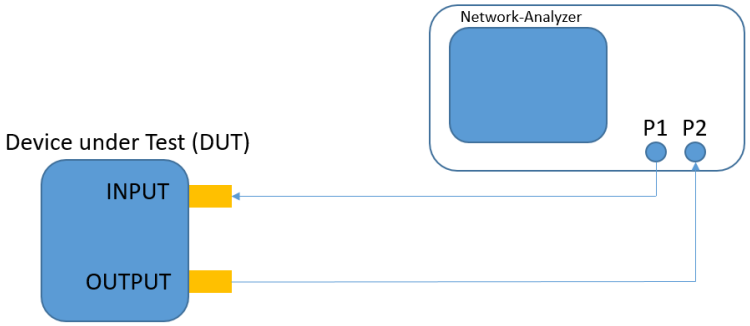

Figure 43: Schematic of a S_{21} measurement.

Figure 44 shows the measurement setup for a S_{12} measurement.

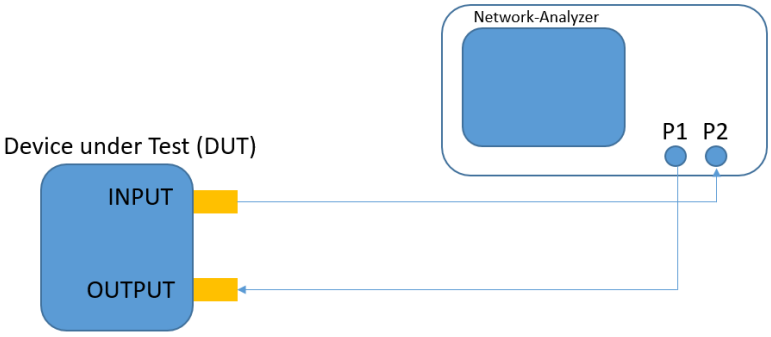

Figure 44: Schematic of a S_{12} measurement.

2.9 Instrumentation of the system

This section is intended to give an overview of the built spectrometer. All components used are explained in more detail in the following sections and the methods used for characterization are explained there.

2.9.1 Setup

Figure 45 shows the setup for the single frequency experiments.

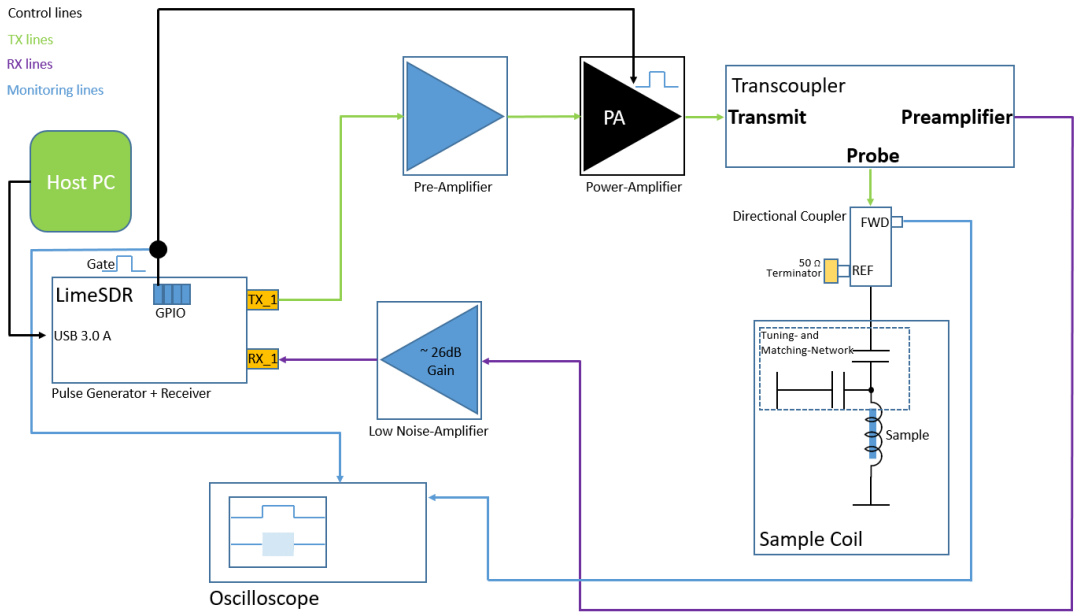

Figure 45: Schematic of the setup for the single frequency experiments.

The SDRs RF-pulses get amplified by the pre-amplifier and the PA and are sent via the transcoupler (the transmit / receive switch, see section 'Transcoupler') to the matched coil. After excitation, the transcoupler switches from TX to RX, which means it connects the coil to the LNA and the SDR receives and digitises the signal. The gate signal is displayed on the oscilloscope. A directional coupler is used to additionally observe the TX signal on the oscilloscope.

2.10 Overview and characterization of the individual assemblies

This section serves to explain the various assemblies used for the realization of the entire spectrometer system (shown in figure 45). Further these were characterized on essential aspects.

2.10.1 Pre-Amplifier

The used pre-amplifier is a BGY587B (NXP semiconductors) which covers a frequency range from $40\,MHz$ up to $550\,MHz$. The Pre-Amplifier would not be absolutely necessary, but if it is used, lower output levels from the SDR are needed and it serves as a protection in case of a reflection on the TX line of the PA that could possibly destroy the TX channel of the SDR. The gain factor according to the datasheet [15] is between $26.2\,dB$ and $27.5\,dB$ across the whole frequency range. The Pre-Amplifier was characterized by measuring the S_{11}, S_{22} and the S_{21} parameter.

2.10.2 Power-Amplifier

The used power amplifier was recycled from an old spectrometer. From previous measurements, it was known that the PA has a maximum output power of around $100\,W$ and can be used in a frequency range from $75\,MHz$ to $140\,MHz$.
For the characterization, the Power-Amplifier's output power and the gain factor were determined by using a custom-built $50\,\Omega$ dummy load. The dummy load has a built-in voltage divider with which a defined fraction of the output voltage can be measured down divided. In order to perform a conversion as accurate as possible, a S_{21} measurement of the division factor of the dummy load was carried out first.
Figure 46 shows the setup to measure the PA's output power. The SDRs TX gain was set to $40\,dB$ and the frequency was set to $100\,MHz$.
The division factor at $100\,MHz$ was determined as $-36.15\,dB$ (see figure 78) and the amplitude of the signal measured from the output of the dummy loads voltage divider is about $1.2\,V$ peak (see figure 79).
The frequency of $100\,MHz$ was used, as this is exactly the $-3\,dB$ bandwidth of the used oscilloscope. It would be necessary to convert the measured peak voltage into the RMS voltage to calculate the RMS-output-power of the PA. As the conversion from peak to RMS voltage is $\cdot\frac{1}{\sqrt{2}}$ and the amplitude of a signal also drops to $\cdot\frac{1}{\sqrt{2}}$ at the $-3\,dB$ bandwidth of the oscilloscope, the displayed peak value can be seen as the RMS value.

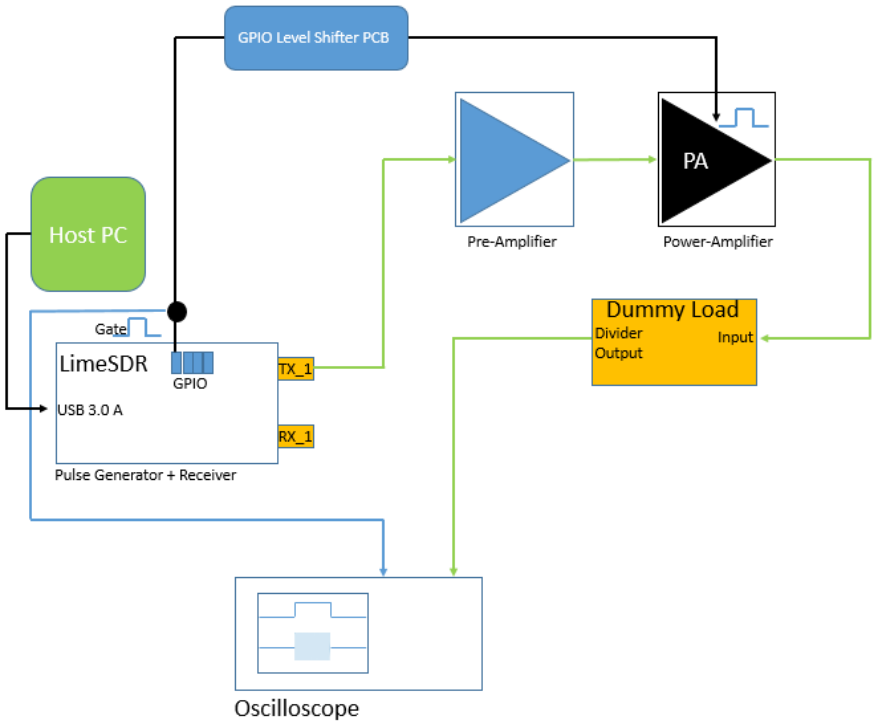

Figure 46: Setup for measuring the PA's output voltage.

The output voltage of the PA can be calculated as:

$$V_{outPA} = 1.2 \cdot 10^{\frac{36.15}{20}} = 1.2 \cdot 64.2 = 77\,V \approx 51\,dBm \tag{42}$$

And the resulting power is calculated as:

$$P_{outPA} = \frac{V_{outPA}^2}{50} = \frac{77^2}{50} \approx 118.6\,W \tag{43}$$

Knowing that the SDRs output voltage at $40\,dB$ gain is $-17.96\,dB$ (see table 2) and the Pre-Amplifier gives a gain of approximately $26\,dB$ (see figure 77) leads to a gain of:

$$G_{PA} = 51\,dBm - (-17.96\,dBm + 26\,dBm) = 42.96\,dB \tag{44}$$

2.10.3 Transcoupler

A transcoupler acts as a transmit / receive switch. In TX mode the RF pulse is transmitted from the PA to the coil, and in RX mode the coil is connected to the LNA. Figure 47 shows the schematic of a standard $\frac{\lambda}{4}$ transcoupler.

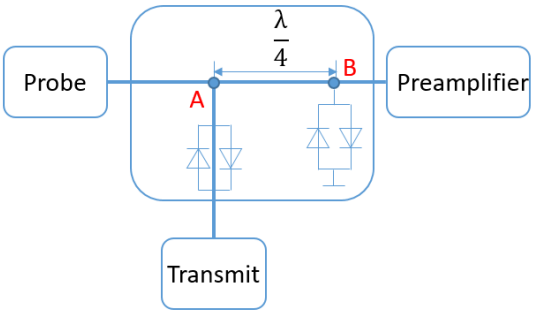

Figure 47: Schematic of a Transcoupler.

A transcoupler uses the fact that a $\frac{\lambda}{4}$ line can transform a short circuit to an open end. For the case, that an RF-pulse is sent from the transmit port, the signal can pass both antiparallel diode pairs. At node B, the diodes act as a short circuit against ground, but this short circuit is transformed to a open end at node A (this means that the preamplifier appears to be disconnected for the TX path). The transcoupler is designed to provide the behaviour of a $\frac{\lambda}{4}$ line over a wide frequency range (here from $80\,MHz$ to $150\,MHz$). The transmission line is then replaced by a special reactive network, the description of which, however, is out of scope of this thesis. If no RF-pulse is sent on the TX channel, there is no ground connection through the diodes at node B. Since the received signals of a sample are low in amplitude (some tens or hundreds of nV to some μV), the diodes in the TX path will block. Since the preamplifier is matched, the length of the line or any wave effects do not matter and thus the sample signal flows into the LNA.

2.10.4 Directional Coupler

A directional coupler can decouple a proportional part of the transmitted and reflected RF signal. In this thesis it is used for monitoring purposes only and is intended to draw attention to (maybe appearing) TX problems immediately. Figure 48 shows a picture of the used directional coupler.

Figure 48: Picture of the used directional coupler.

A more detailed documentation on this device can be found in [16].

2.10.5 Sample Coil

The sample coil is equipped with two variable capacitors and mounted in a shielded box. Figure 49 shows a picture of the used sample coil.

Figure 49: Picture of the used sample coil including the Tuning and Matching Capacitors.

It is of high importance to tune and match the coil at the frequency to be used. Therefore, a Network Analyzer (NWA) is connected to the terminal at the frame of the coil. An S_{11} measurement is then performed and the coil is tuned and matched using the two capacitors. Figure 50 shows the setup for an S_{11} measurement of the sample coil (see also figure 41, this figure was intentionally redrawn with the coil because this measurement must be performed before any measurement of a NQR-sample and is therefore very important).

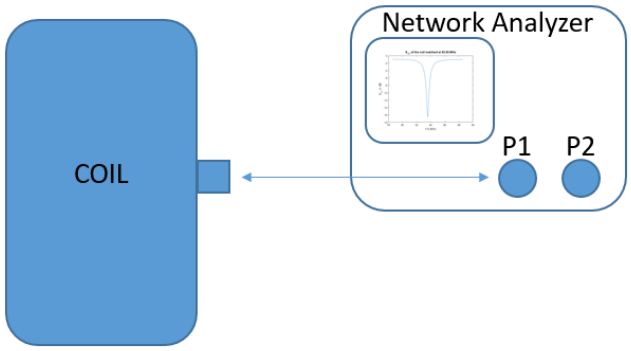

Figure 50: Schematic of the S_{11} measurement of the sample coil.

2.10.6 Low Noise Amplifier

An LNA is needed to amplify the received signals above the noise floor of the spectrometers RX port. The used LNA is a SPF5189Z evaluation board with a frequency range from $50\,MHz$ to $4\,GHz$. Additionally, the SPF5189Z has a ultra-low noise figure of $0.6\,dB$ specified at $900\,MHz$. Figure 51 shows a picture of the used LNA.

Figure 51: Picture of the SPF5189Z Low Noise Amplifier.

Since the LNA offers a very wide frequency range and the lowest part is used here, the S-parameters were measured in the used frequency range. In the data sheet [14], one can find the specifications for the amplifier around the middle of its application range, to be precise for the range around $2\,GHz$. The LNA was characterized by measuring the S-Parameters S_{11}, S_{22} and S_{21}. For the following noise analysis, two of the LNAs were used in series. Therefore a S_{21} of two SPF5189Z in series connection was recorded too.

2.11 Pulse Sequences

This section is intended to provide information about the pulse sequences used in this work. SNR characterizations were then performed using these pulse sequences. As already mentioned in the functionality check section, all sequences presented here were also looped back into the RX path via a $30\,dB$ attenuator to show the time course in the base-band (see results from figure 70 to figure 73).

2.11.1 Free Induction Decay

First, an optimal timing for the Free Induction Decay (FID) was determined by optimizing the pulse length for the powder sample of $BiPh_3$ which was used as a reference sample, and afterwards the timing for the Spin-Echo sequences was set. The optimal pulse length for the FID, corresponding to a 90° pulse, was about $3\,\mu s$. Further, a blanking time of $7\,\mu s$ was used. The total acquisition time was (as already mentioned in section 'Signal Processing') set to about $398.4\,\mu s$

and the window length was set to $20\mu s$. The acquisition time and window length parameters were kept the same for comparison with other sequences. For all of the sequences presented here, an offset of about $13\mu s$ was incorporated. This has no particular reason, but it leads to a better visibility of the pulse. The repetition time was set to $5\,ms$ for all sequences. The settings for the SDR system can be looked up in table 5.
Figure 52 shows the timing diagram for the FID.

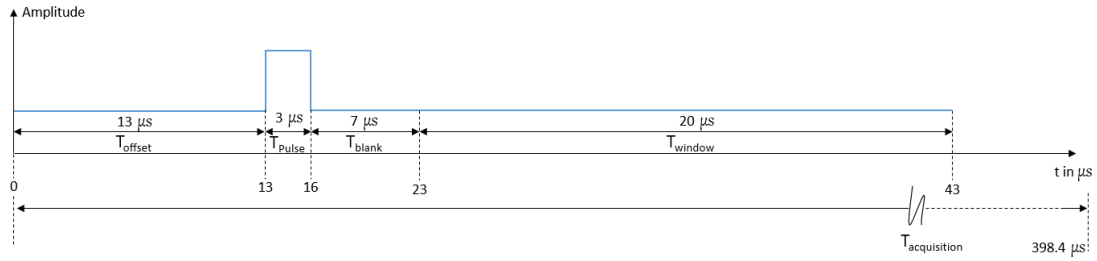

Figure 52: Timing diagram of the FID sequence ($T_R = 5\,ms$).

Screenshots of the python script for this sequence can be found in the appendix (figures 125 and 126).

2.11.2 Spin-Echo

For the Spin-Echo, a 90° and a 180° pulse was used, using the pulse length from the previous FID and twice the pulse length, respectively. Further, an echo time (T_E) of $21\,\mu s$ was used.

Figure 53 shows the timing diagram for the Spin-Echo.

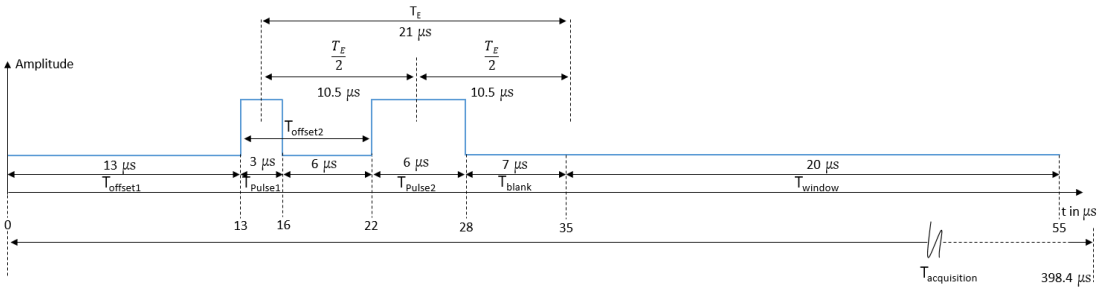

Figure 53: Timing diagram of the Spin-Echo sequence ($T_R = 5\,ms$).

Typically, the window length (T_W) is centered around the echo. Here, however, a very short echo time was used and only about half of the echo was recorded to avoid the main part of ringdown artefacts. The short echo time was chosen on the one hand because similar high signal peaks were reached as compared to the FID sequence, on the other hand it was the idea to also show a comparison between

ringdown suppressing and normal sequences using only one measurement. Most of the ringdown should be decayed during a blanking time of $6-7\mu s$, but differences in the SNR should be visible, since some residues always remain. The echo time was chosen in a way that the expected echo reaches its maximum after $7.5\,\mu s$ after the π pulse. Thus, the same blanking as for the FID sequence ($7\,\mu s$) can be used. Note: usually there is no blanking time with a Spin-Echo sequence, but the time between the last pulse and the start of the acquisition window was named T_{blank} for all sequences to compare them among each other.
Screenshots of the python script for this sequence can be found in the appendix (figures 127 and 128).

2.11.3 Spin-Echo with Phase-Cycling

Phase cycling is a method for removing artefacts that occur synchronously with the RF-pulse. The phase of the RF pulses and the acquisition reference phase are cycled over a certain number of repetitions (typically 4 or 8) with a special pattern. The phasing is chosen in such a way that artifacts sum up destructively during averaging while the spin signals sum up constructively. For the phase-cycling presented here and the composite pulse presented in the following section, please also refer to [21].

A known problem is the ringdown of the sample coil. The RF-pulses excite the resonator and thus the voltage across the coil decays after a pulse is over. A particular problem with ringdown is that it occurs at the same frequency as the expected signal from the sample. Due to the high power of the pulses, the ringdown signal has a much higher amplitude than the sample's signal. However, most of the ringdown has almost completely decayed after a few μs. Without suppression of this artifact, the ringdown must be blanked, which can lead to the signal no longer being measurable at short T_2 or T_2* times. The phase-cycling scheme used here is called Kazan Echo. Table 3 was taken over from [19] and shows the Kazan Echos phase constellations.

Table 3: Cycles of the Kazan Echo.

Kazan Echo			
Cycle	Phase 90° ($\frac{\pi}{2}$) pulse	Phase 180° (π) pulse	Acquisition Phase
1	0	1	0
2	1	1	3
3	2	1	2
4	3	1	1

The phases 0°, 90°, 180°, and 270° are abbreviated by the numbers $0-3$. That means, $0° = 0$, $90° = 1$, $180° = 2$, and $270° = 3$. For a more comprehensible representation figure 54 shows the pulse trains 1 and 3 of the Kazan Echo. The coordinate systems used here rotate at the Larmor frequency (around the z-axis). The following explanations have been translated and supplemented or slightly modified by [19].

(1a) The magnetic field (B1) generated by the 90° RF pulse tilts the magnetization (M) into the xy-plane. The phase shift of the 90° pulse for Kazan-Pulse 1 is 0°, which corresponds exactly to the respective phasor direction along the positive x-direction in the illustration.

(1b) Due to B_0 inhomogeneities the spins get out of phase (here representative only one phasor (M) for a single isochromate). The magnetic field changes cause spins to precess with a slightly different velocity than the coordinate system (angle ϕ).

(1c) The second pulse of the echo sequence (here again labeled B1) is a 180° pulse, which has a phase shift of 90° (therefore the B1 phasor is rotated by 90° compared to (1a)). The 180° pulse mirrors the spins (here again the representative isochromate M) about the y-axis and the spins run towards each other until they overlap again (rephasing). Exactly after the echo time all isochromates are again exactly superimposed on the y-axis.

(1d) The acquisition is performed without phase shift (i.e. 0°, the corresponding phasor points in positive x-direction). The ringdown artifacts always occur with the same phase as the transmitted pulse, here with the phase of the 180° pulse (assuming the ringdown of the 90° pulse has already decayed). This means that the ringdown also has the phase of 90°, i.e. the same as the spin signal.

(2a) The 90° pulse (B1) at Kazan pulse 3 has a 180° phase shift, which is why the magnetization (M) is exactly tilted to the negative y-axis.

(2b) Again, dephasing of the spins occurs.

(2c) Due to the 180° pulse (B1) the spins rephase on the negative y-Axis.

(2d) The acquisition phase this time is 180° (which corresponds to a phasor along the negative x-direction). Since the ringdown has is in phase with

the 180° pulse (which always has a 90° phase shift in Kazan Echo), the ringdown again has a phase of 90°.

If the signals obtained in this way are subtracted, the signals superimpose constructively and the ringdown destructively. This process must still be carried out for the two other pulses (2 and 4), whereby these then also superimpose constructively with respect to the signal and destructively with respect to the ringdown.

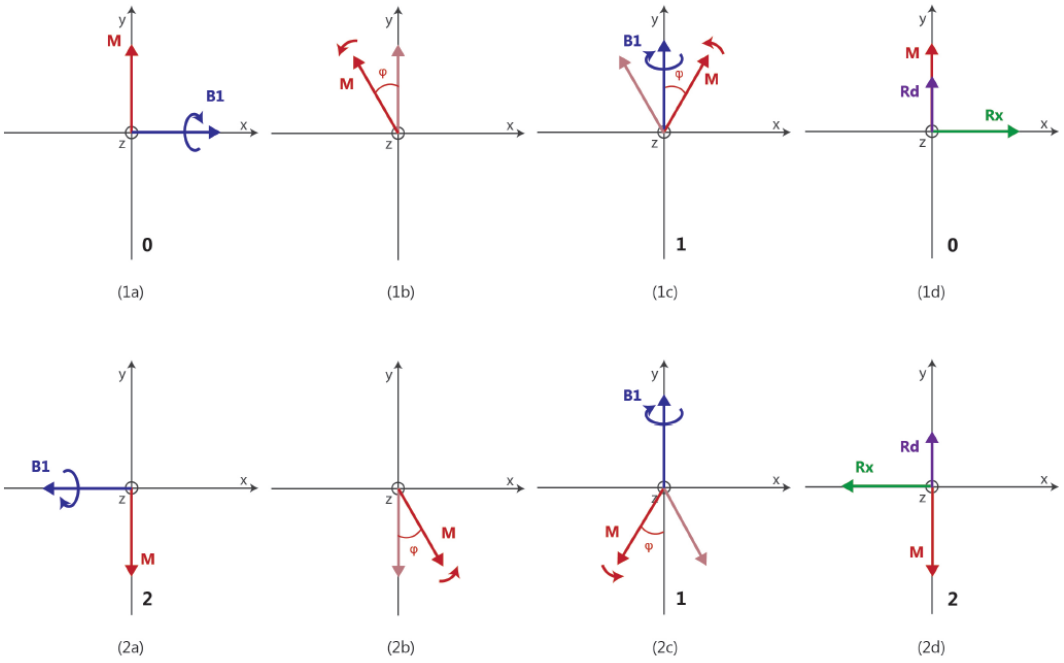

Figure 54: Phase diagram for phases 1 (upper row) and 3 (lower row) of the Kazan echo. Image Source: [19].

The timing of the phase-cycled Spin-Echo sequence used, was set the same as it was set for the non-phase-cycled Spin-Echo in figure 53. Screenshots of the python script for this sequence can be found in the appendix (figures 129 and 130).

2.11.4 Composite Pulse

A composite pulse consists of a pulse which changes its phase with time, i.e. undergoes phase modulation. As with the Kazan phase-cycling presented above, the Composite Pulse experiment is also performed four times with different phase shifts and is also used for suppressing coil ringdown and other artefacts synchronous to the RF-pulse.

Table 4 shows the Composite Pulse phase constellations and was taken over from [20]. In this case the phase changes discontinuously after a certain time within the pulse.

Table 4: Composite Pulse phase constellations. Source: [20].

Composite Pulse					
FID	Phase first part	Phase second part	weighting	sign of ringing 1^{st} pulse	sign of ringing 2^{nd} pulse
1	0°	45°	−1	+	+
2	0°	135°	+1	+	+
3	0°	−135°	−1	+	−
4	0°	−45°	+1	+	−

According to [20], the first and second partial pulses have phases $(0, x)$ and $x = 45°$ and the condition to be satisfied for the pulse of the sequence is $(0, x)$, $(0, -x + 180°)$, $(0, x + 180°)$, $(0, -x)$.

In [21], first experience with Composite Pulses is documented. On their recommendation also in this work $x = 45°$ was chosen and the lengths of the partial pulses were set to $3\,\mu s$ and $6\,\mu s$ corresponding to flip angles of $\frac{\pi}{2}$ and π (see FID).

Figure 55 shows the timing diagram for the Composite-Pulse.

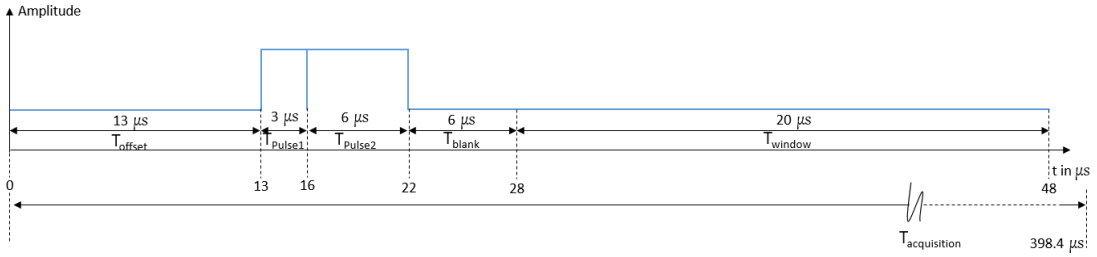

Figure 55: Timing diagram of the Composite-Pulse sequence ($T_R = 5\,ms$).

Note: Between the last pulse and the start of the readout, a blanking of about $7\,\mu s$ was used for the FID and the Echo sequences (see figure 52 and figure 53). For the Composite Pulse, only $6\,\mu s$ was used due to a mistake during the pulse programming. Since the composite pulse suppresses ringdown very well, this makes almost no difference with respect to the SNR.
Screenshots of the python script for this sequence can be found in the appendix (figures 131 and 132).

2.11.5 Settings of the SDR based system

Table 5 shows the settings for the sequences of the SDR based spectrometer. The settings above the first double line represent the settings comparable to the SCOUT and other spectrometers, the settings below the first double line are SDR specific. The settings that adjust the IQ mixers have been optimized for about $83.8\,MHz$, but they are still calibrated at $90\,MHz$ (see also appendix for the calibration routine). If the system is used for other frequency ranges, they may need to be adjusted. Further, the phase settings for the Phase Cycled Spin-Echo sequence and the Composite Pulse can be read below the second double line.

Table 5: SDR sequence settings.

SDR settings	
Setting	Value
LO Fequency (l.lof)	$82.36\,MHz$
IF Fequency (l.pfr)	$1.2\,MHz$
Sample Rate (l.sra)	$30.72\,MS/s$
Number of Averages (l.nav)	1000
Repetition Time (l.trp)	$5\,ms$
TX lowpass (l.tlp)	$130\,MHz$
RX lowpass (l.rlp)	$3\,MHz$
TX Gain (l.tgn)	$40\,dB$
RX Gain (l.tgn)	$55\,dB$
Acquisition window	$20\,\mu s$
Not initialize (l.noi)	-1
TX I DC correction (l.tdi)	-45
TX Q DC correction (l.tdq)	-0
TX I Gain correction (l.tgi)	2047
TX Q Gain correction (l.tgq)	2039
TX phase adjustment (l.tpc)	3
RX I DC correction (l.rdi)	0
RX Q DC correction (l.rdq)	0
RX I Gain correction (l.rgi)	2047
RX Q Gain correction (l.rgq)	2047
RX phase adjustment (l.rpc)	1
GPIO Trigger 3 (l.t3d)	[1, 0, 50, 10]
Total Acquisition Time (l.tac)	$82\,\mu s$ selected, which result to $398\mu s$ due to min. buffer size and default multiplication of buffer size $\cdot 3$
Phase Cycled Spin-Echo Phase Constellation (l.pcn)	[4, 1]
Phase Cycled Spin-Echo Phase (l.pph)	[0, np.pi/2]
Composite Pulse Phase Constellation (l.pcn)	[1, 4]
Composite Pulse const. Phase (l.pph)	[0, np.pi/4]

Using the sample rate of $30.72\,MS/s$ and the chosen window length of $20\,\mu s$ results in about $30.72 \cdot 10^6 \cdot 20 \cdot 10^{-6} \approx 614$ data points.

2.12 Noise Analysis

2.12.1 Conversion factor between spectrometer unit and volt

It is first necessary to establish a conversion factor between the arbitrary unit of the spectrometer and the true signal strength in volts. For the determination of the conversion factor it is useful to generate a reference signal with exactly known amplitude with the SDR and to connect it directly to the RX port. It should be noted that such measurements should be carried out with small amplitudes, otherwise the RX port may be damaged. For such measurements, attenuators with $20\,dB$ or even $30\,dB$ can be added for safety, as the input of the SDR is very sensitive.

In this work, the SDRs TX port was connected to the RX port via a $30\,dB$ attenuator. A sequence with a single pulse (FID) with a TX gain of $40\,dB$ and a frequency of about $83.5\,MHz$ was used. A TX gain of $40\,dB$ leads to a peak voltage of about $40\,mV$ (see table 2). Python was then used to calculate the RMS value of the real part of the received time domain signal (in the arbitrary spectrometer unit), which was about 1450.84 a.u.. The signal was recorded with a number of 1000 averages, which have already been taken into account here (time domain data was divided by the number of averages already). The above mentioned internal RX-Gain was set to $55\,dB$. Figure 85 shows the recorded time domain data.
The frequency was chosen because the reference measurements that follow later are based on $BiPh_3$ and this substance has a transition at $83.5\,MHz$.
Figure 56 shows the setup for determining the conversion factor from a.u. to volts (same as for the loopback check, figure 35).

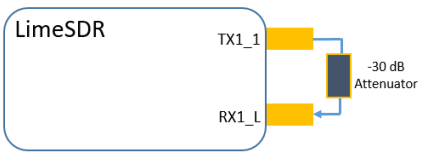

Figure 56: Schematic of the setup for the conversion factor from a.u. to volts.

The conversion factor can then be calculated as:

$$scaling\_factor = \frac{1450.84}{(\frac{40 \cdot 10^{-3}}{10^{\frac{30}{20}} \cdot \sqrt{2}})} = 1622137.746 \tag{45}$$

The equation above takes the conversion of the peak value to RMS and the $30\,dB$ attenuator into account. This means a signal with $1\,V_{RMS}$ would lead to 1622137.746 a.u..
It is important to keep in mind, that the RX-Gain has to be taken into account if it is not set to $55\,dB$ and a scaling of the spectra or time signals is done. However, this can be corrected by calculating the difference to $55\,dB$ and taking it into account as additional factor.
Screenshots of the python script for evaluating the scaling factor can be found in the appendix (figures 133 and 134).

2.12.2 Theoretical vs. measured noise

The goal of the noise analysis is to verify that the system has a low noise figure, i.e. the noise seen in the data is as close as possible at the theoretical noise of a $50\,\Omega$ resistor.
The concept behind the noise analysis performed here is to connect a $50\,\Omega$ resistor to the RX port and amplify the noise caused by it above the noise level of the RX port. Two LNAs are used in series here to ensure sufficient amplification. In order to carry out an accurate investigation, an S_{21} measurement of the amplifier chain was recorded. The gain of the LNAs adds up with the gain of the SDR internal RX amplifiers. As for the determination of the conversion factor, the RX-Gain was set to $55\,dB$. It is not necessary to know the exact gain of the internal amplifiers for analyzing the noise, as the RX-Gain factor is included (but hidden) in the conversion factor. Also important for this measurement was the RX-lowpass filter which was set to $3\,MHz$ and the acquisition time of $20\mu s$ was again used. To perform the analysis, a sequence can again be started by a python script. Here it is again important to set an appropriately high blanking time in order to really only measure noise (here, $20\,\mu s$ blanking time was used, which is $13\,\mu s$ more compared to the blanking used with the FID sequence in figure 52). The noise in the time domain can be calculated using the standard deviation from the real part of the complex RX time domain data. The standard deviation was determined with the help of python. This measurement was also carried out with 1000 averages.

Figure 57 shows the setup for the Noise Analysis.

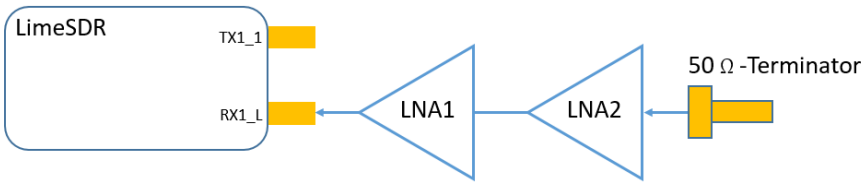

Figure 57: Schematic of the setup for the Noise Analysis.

The gain of the amplifier chain at 83.5 MHz is about 52 dB (see figure 84). Again, the range around 83.5 MHz was examined, as this is close to the measuring range used later.

The theoretical noise of a 50 Ω resistor is calculated as follows:

$$\sigma_N = \sqrt{4kTBR} = \sqrt{4 \cdot 1.38 \cdot 10^{-23} \cdot 293.15 \cdot 3 \cdot 10^6 \cdot 50} = 1.558\,\mu V \tag{46}$$

Since the data has been averaged and a comparison between theoretical and measured noise is done, the number of averages must be taken into account. The noise in time domain decreases with $\sqrt{averages}$, resulting in an actual theoretical noise at 1000 averages:

$$\frac{\sigma_N}{\sqrt{1000}} = \frac{1.558 \cdot 10^{-6}}{\sqrt{1000}} = 49.26\,nV \tag{47}$$

The measured standard deviation (already divided by 1000 because of the averages) was 33.89 a.u. (see figure 86). The measured noise in volts can then be calculated as:

$$\frac{a.u.}{LNA\_Gain \cdot scaling\_factor} = \frac{33.89}{10^{\frac{52}{20}} \cdot 1622137.746} = 52.479\,nV \tag{48}$$

Screenshots of the python script for the noise analysis can be found in the appendix (figures 135 and 136).

2.13 SCOUT system

2.13.1 Setup

In order to compare the performance with respect to the Signal to Noise Ratio (SNR) of the LimeSDR based spectrometer with a commercially available one, reference measurements with the same sample ($BiPh_3$) were carried out using the SCOUT spectrometer. In order to achieve a fair comparison, the same components were used wherever possible. The following components were the same: Transcoupler, Low Noise Amplifier, Sample and sample coil. Unfortunately, it was not possible to use the same power amplifier because the scout setup was needed for experiments at a different location. In this respect, the two spectrometers could not be compared under the same environmental conditions. The power amplifier used in the SCOUT setup is a Tomco BT00500-GammaS $5\,MHz - 310\,MHz$ with a maximum power of $500\,W$. The comparison was made with the same pulse settings as above (figures 52, 53, 55). Care has been taken to ensure that the power is also around $100\,W$, but minor changes were tolerated for optimum pulses.

Figure 58 shows the setup for the reference measurements with the SCOUT spectrometer.

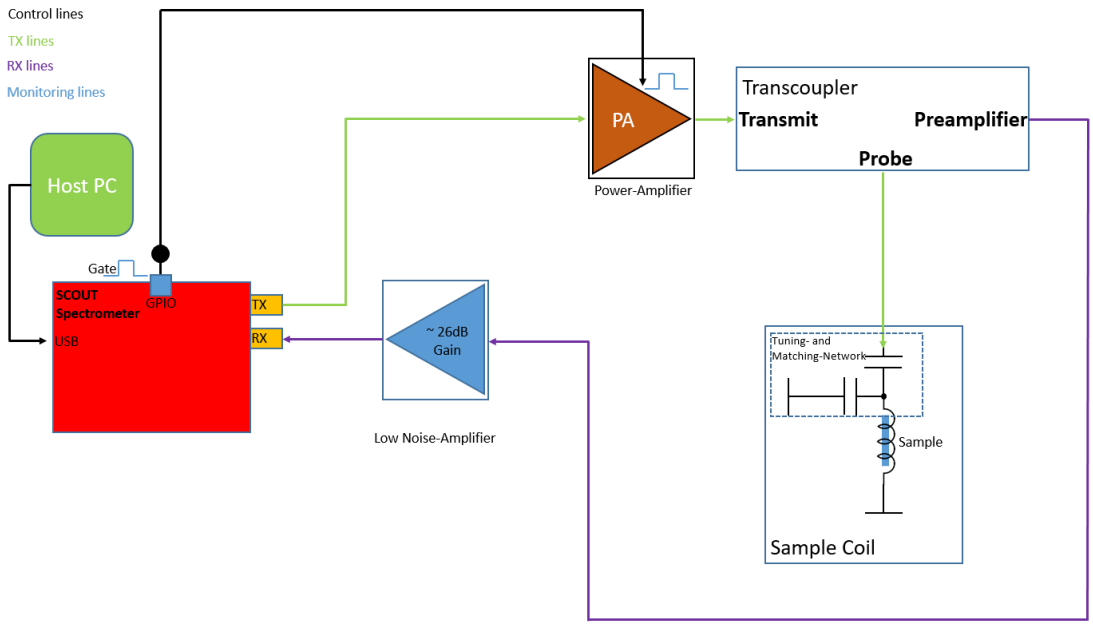

Figure 58: Setup for the reference measurements with the SCOUT spectrometer.

2.13.2 Settings SCOUT system

Table 6 shows the settings for the SCOUT spectrometer.

Table 6: SCOUT settings.

SCOUT settings	
Setting	Value
Observ. Freq	$83.56\,MHz$
Acq. Points	330
Points 1D	512
SW +/-	$8.33\,MHz$
Filter	$8.33\,MHz$
Dwell Time	$60\,ns$ (sampling rate of $16.66\,MHz$)
Acq. Time	$19.8\,\mu s$
Last Delay	$5\,ms$
Scans 1D	1000
Actual Scans 1D	1000
Receiver Gain	300
TX Gain	25 (leads to approx. $100\,W$)

The settings in Table 6 were applied for all sequences. The reference measurements were also carried out with the four sequences FID, Spin-Echo, Spin-Echo with Phase-Cycling and Composite Pulse with the same timing as shown in figures 52, 53 and 55.

2.14 Signal to Noise Ratio (SNR) determination

The measurements with the sequences described above, were always carried out twice. First, with the sample located in the sample coil and second without the sample to determine the noise. Therefore, the coil was always tuned and matched for both cases respectively. As the experiments were carried out at only one frequency, the standard deviation of the noise measurements were always calculated using approximately $\pm 500\,kHz$ of the data with respect to the peak recorded with the sample. The SNR in frequency domain is then calculated as:

$$SNR = \frac{max(Peak)}{std(noise_{1\,MHz-without-sample})} \tag{49}$$

2.15 Conducted Measurements

This section refers to measurements done with the system according to figure 45 and figure 58. These are single frequency experiments, this means the samples were only excited at the target frequency.

2.15.1 Reference Measurements between SDR based system and SCOUT system

The elaborated sequences (section 'Pulse Sequences') were tested on $BiPh_3$ with the SDR based system according to figure 45 and also tested with a commercially available spectrometer (SCOUT, Tecmag) for comparison. The SCOUT system can be seen in figure 58. The spectrometer shown in figure 45 covers a frequency range from about $80\,MHz$ to $140\,MHz$, which is limited downward by the transcoupler used and upward by the PA. $BiPh_3$ was used because this substance has a transition at about $83.5\,MHz$ and is known to give comparatively high signals in the single-digit μV range from previous measurements (see also [17]). $BiPh_3$ further has the following time constants at the transition at $83.5\,MHz$: $T_1 = 820\,\mu s$, $T_2 = 396\,\mu s$ and $T_2* = 50\,\mu s$. Data for T_1 and T_2 are from an unpublished table at the institute, T_2* was estimated from the time domain using FID and a long window length ($T_{window} = 175\,\mu s$). Measurements with $BiPh_3$ were recorded with identical sequences on both systems with 1000 averages. The pulse frequency was set to around $83.56\,MHz$, with minor changes due to warming up of the sample which leads to resonance shifts. Results can be seen in the figures from 87 to figure 94.

2.15.2 Further measurements with the SDR spectrometer

To show the effectiveness of the artefact cancellation of the spin-echo with phase-cycling and the composite pulse more clearly, extra measurements were performed (again with $BiPh_3$). The blanking times were shortened to $4\,\mu s$ for all four sequences (FID, spin-echo, spin-echo with phase-cycling and composite pulse) after the last pulse until the beginning of the acquisition (see figures 52, 53 and 55, the time interval T_{blank} was set to $T_{blank} = 4\,\mu s$ for all four sequences). This means only the acquisition window was shifted but it was left the same length ($T_{window} = 20\,\mu s$) for all four sequences. Results can be seen in figure 95.
For showing the system's performance at low SNR also samples with known weak signals were measured for comparison. These samples were Bi_2O_3 powder (ReagentPlus ®, powder, $10\,\mu m$, 99.9 % trace metals basis, SigmaAldrich) and a modification obtained by grounding for 30 min with 600 rpm for reducing the crystallite size. This substances were then measured using the same sequences as presented in section 'Pulse Sequences'. The relaxation times of Bi_2O_3 are not known exactly in either the unground or ground states, but at most T_2 and especially T_2* are estimated to be a factor of $5-10$ shorter than for $BiPh_3$. The transition frequency of Bi_2O_3 is $90.4\,MHz$. Here 1000 or even 10000 averages were used, as the ground Bi_2O_3 is very difficult to measure without exhaustive

averaging. This substance was chosen because its signal could not be detected in the 'CONCRADLE' system (mentioned in the introduction).

2.16 Concept for the usage of broadband probe-heads for broadband-scans

In this section a concept is presented, with which it is possible to use the broadband probe-heads available at the Institute of Biomedical Imaging with the SDR spectrometer. The methods explained here are intended to give an understanding of the assemblies and are, however, to be seen as a prototypical concept and only to prove the basic feasibility.

2.16.1 Introduction to Broadband probe-heads

In recent years, several broadband probe-heads were built for different frequency ranges at the Institute of Biomedical Imaging. However, the principle of the TX/RX switching and the matching strategy is the same for all of those. In this work, a probe-head (internal name: CRYO65 − 120GEN-3A) designed for the range between $65 - 120\,MHz$ was used.
Figure 59 shows the principle of the broadband probe-heads.

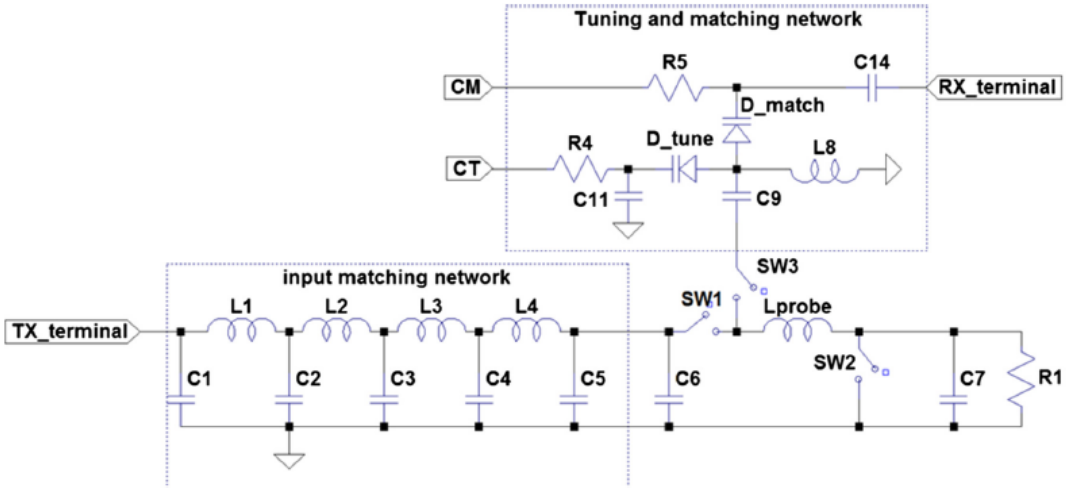

Figure 59: Principle of the TX/RX switching and Matching principle of the broadband probe-heads, Source: [17]. The actual circuit is more complex, parts have been omitted here for simplicity.

The idea is to split the resonator into a static broadband matched TX path (S_{11} over several tens of MHz $\leq -15\,dB$ to $-12\,dB$) and a variable narrowband matched RX path. For the RX path, care should be taken that $S_{11} \leq -20\,dB$ at any frequency as stronger reflections degrade the SNR. The main advantage of this strategy is that the very small RX signals can be received with high

quality factors than at TX, where low Q-factors can be accepted for the sake of large bandwidth. Further, fast scanning of broad spectra is possible, since the varactor diodes can tune and match the resonator very quickly when setting the corresponding reverse bias voltages. In the TX path, it is not necessary to have a matched configuration of $S_{11} \leq -20\,dB$, since most RF power amplifiers accept a S_{11} parameter of $S_{11} \leq -8$ to $-10\,dB$.
The TX path contains an exponential matching line (with a number of minimum three and up to five LC elements), realizing broadband matching over tens of MHz. Here, the power-resistor R_1 is transformed to approximately $50\,\Omega$ with the capacitor C_7, the sample coil and the exponential matching line. In the RX path, the matching network is provided by the sample coil and by variable capacitance diodes (varactor diodes). The capacitance of these varactor diodes can be adjusted by applying a reverse bias voltage. The TX path is separated from the RX path by switches consisting of PIN diodes. During TX, switch SW1 is closed and the other two switches SW2 and SW3 are open. In RX SW1 is open and SW2 and SW3 are closed [17]. The opening and closing of these switches is controlled by applying certain voltages to the respective PIN diodes. The switching is controlled by the Logic for Timing, Control and Protection explained in the following section.
A broadband S_{11} measurement of the TX path can be seen in figure 104 and, as an example, S_{11} measurements were recorded at different frequencies in the RX path (figure 106).

2.16.2 Logic for Timing, Control and Protection (LOPRO)

The Logic for Timing, Control and Protection (LOPRO) is responsible for the switching between TX and RX. Further it puts the probe-head into the tuning and matching configuration if no gate signal is present at port 'P'.
Figure 60 shows the timing diagram of the LOPRO ports used in this work. The timing was looked up from [18]. The LOPRO has some additional outputs and functionalities which were not used in this work (see also [18]).

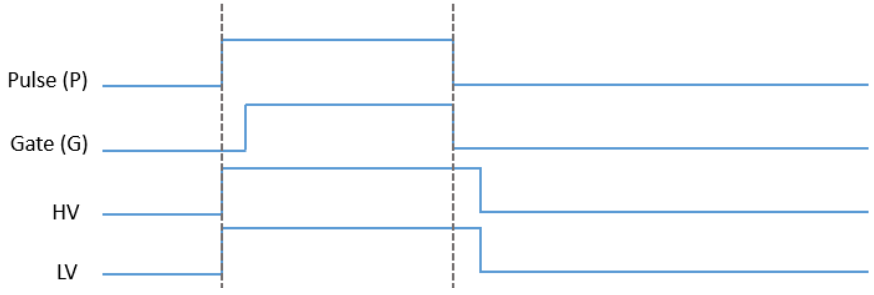

Figure 60: Timing diagram of the LOPRO.

The gate signal, which is output synchronously to the RF-pulse from the SDR, is connected to the port 'pulse' (input) of the LOPRO. The gate port of the LOPRO (output) serves as the gate for the power amplifier and is output with a safety delay of $5\,\mu s$ relative to the original gate signal (see Results). The safety delay is adjustable by RC Elements present in the LOPRO. Note: this adjustment function was realized afterwards and is not implemented in the original work from [18]. The LV (stands for Low Voltage) and HV (stands for High Voltage) signals are used for switching the PIN diodes indicated in figure 59. The LV defaults to $-5\,V$ and switches to $+5\,V$ when a signal turns high at the 'Pulse' port. The HV defaults to $-5\,V$ and switches to $350\,V$ when a signal is applied to the 'Pulse' port.

In principle, a pin diode can be seen as a switch which is in the 'closed' state when a forward bias greater than the forward voltage is applied across it. To achieve high isolation in the 'open' state, a high negative voltage is applied. This high isolation in PIN diodes is achieved by the intrinsic layer between the p- and n-doped regions.

The PIN diodes, indicated by switches SW2 and SW3, are reverse biased in TX mode to $350\,V$. This means that they are in a high isolating state and the switches SW2 and SW3 can be considered as 'open'. The PIN diodes indicated by the switch SW1 (which in some implementations are replaced by antiparallel Si diode pairs) are not reverse biased in TX mode so that the RF-pulse can pass. The RF-pulse can flow through the exponential matching line to the sample coil and the capacitor C_7 and resistor R_1 (figure 59). In RX mode the PIN diodes indicated by switches SW2 and SW3 are forward biased with $5\,V$, which means they can be seen as closed switches. The RX signal cannot pass cannot pass SW1. If SW1 is implemented with a PIN diode, it is reverse biased and thus in closed state. In case of an implementation as antiparallel Si-diode pair, the RX signal is too small to reach the threshold voltage for diode opening. Thus, SW1 can be seen as an open switch. The RX signal can then flow through the coupling capacitor C_9 to the RX port via the RX matching network.
By taking a closer look into figure 60 it can be seen that HV and LV are switched simultaneously. The delay shown in figure 60 is also present when comparing the rising edges of the HV/LV signals with the rising edge of the gate signal. It can also be observed, that a similar delay is present between the falling edge of the gate signal and the falling edge of the HV/LV signals.
The delays serve primarily for the protection of the sensitive varactor diodes and the LNA. The first delay makes sure that the probe-head is in TX mode before the high power RF-pulse is applied, so that the RX path is not damaged. Conversely, the second delay is necessary in order to safely switch back to RX mode after

the RF-pulse is over. An oscilloscope was used to record the gate signal from the SDR, the resulting gate signal from the LOPRO, and the resulting RF-pulse (see figure 105).

2.16.3 Automated tuning and matching for broadband probe-heads

Figure 61 shows the schematic of the setup for the Tuning- and Matching process.

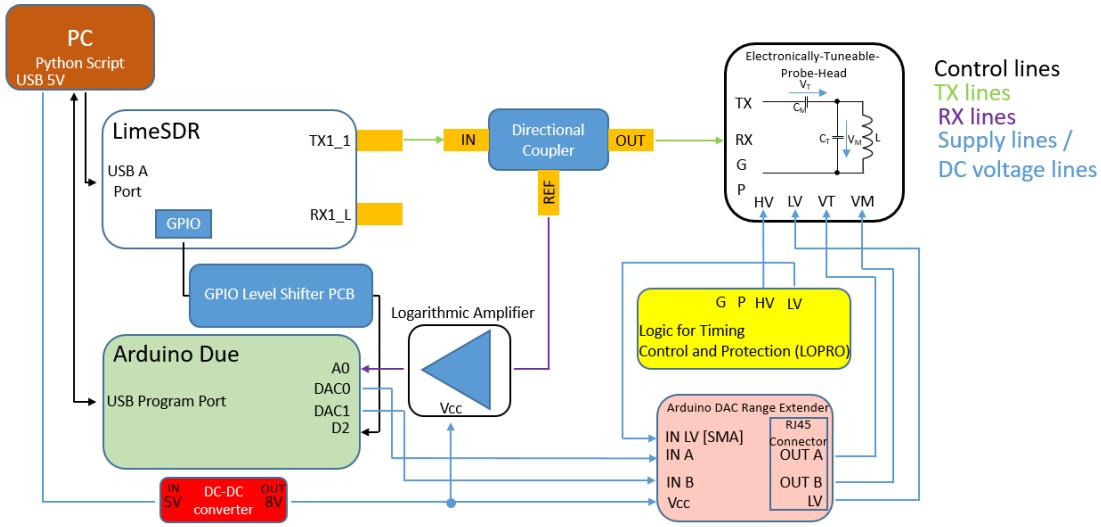

Figure 61: Schematic of the setup for the Tuning- and Matching process.

2.16.4 Automated tuning and matching routine

The setup in figure 61 above applies RF pulses with a certain target frequency (duration about $200\,\mu s$ at a repetition time of about $400\,\mu s$) via a directional coupler (Minicircuits 15542, ZFDC-20-5) to the probe-head's RX port. The SDR outputs a gate signal synchronous to the RF pulse, which is connected to the 'Arduino Due'. The matching network consisting of two varactor diodes and the sample coil can be matched to the currently applied frequency by changing the voltages applied to the respective varactor diodes.
Beforehand, a subdivision of the range of tuning and matching voltages is chosen. The permissible voltage range of the varactor diodes is approximately $0-5\,V$ and this is typically divided into $50-100$ steps. These voltage steps are then tried in all combinations for the tuning varactor diode and the matching varactor diode, and the reflected wave for each combination is then fed via the directional coupler to a logarithmic amplifier (AD8310, Analog Devices). The AD8310 is connected to a analog input of the 'Arduino Due' where the signals is digitised. The logarithmic amplifier converts the RF signal to its envelope, hence it yields a constant output voltage depending on the voltage level of the RF signal (see

also [16] for a detailed explanation and characterization of the AD8310). The microcontroller starts to record the reflected wave as soon as the gate signal is detected as high. The data points obtained in this way are averaged (note: the build-in AnalogRead() function of the Arduino library was used, which needs about $100\,\mu s$ for one conversion therefore there are about two averages per cycle). The combination of the tuning and matching voltages which yield the lowest reflection is stored in a lookup table together with the corresponding frequency. This routine is performed over the entire frequency range under investigation in steps of typically $100\,kHz$.
It should be noted that the choice of voltage steps has a large impact on the runtime, as U_{step}^2 combinations are tried per frequency ('exhaustive search'). The control of the SDR and the 'Arduino Due' work via USB and the commands for both devices run via a python script. Using python, a lookup-table is created which contains the tuning and matching voltages for optimum matching and the corresponding frequency. During an actual NQR scan, the corresponding voltages can be set for each excitation frequency.
It was necessary to adjust the output voltage range of the 'Arduino Due's' DACs. Out of scratch, the two on board DACs of this microcontroller only have an output voltage range of $0.55 - 2.75V$. The PCB, which is the light red block in figure 61, was realized in a different project (Master Seminar, see [23]) and contains a differential amplifier for each of the DACs which increases the output voltage range to about $0 - 5\,V$. For the supply of this board and the logarithmic amplifier, a DCDC converter was connected to another USB socket from the host PC which converts the usual $5\,V$ to $8\,V$.

Note: when looking at figure 61 it can be seen that no gate signal is connected from SDR to LOPRO (yellow). This is important because in this way the probehead is in tuning and matching mode.

Figure 62 shows the flow chart of the tuning and matching routine for both, the python script and the 'Arduino Due'.

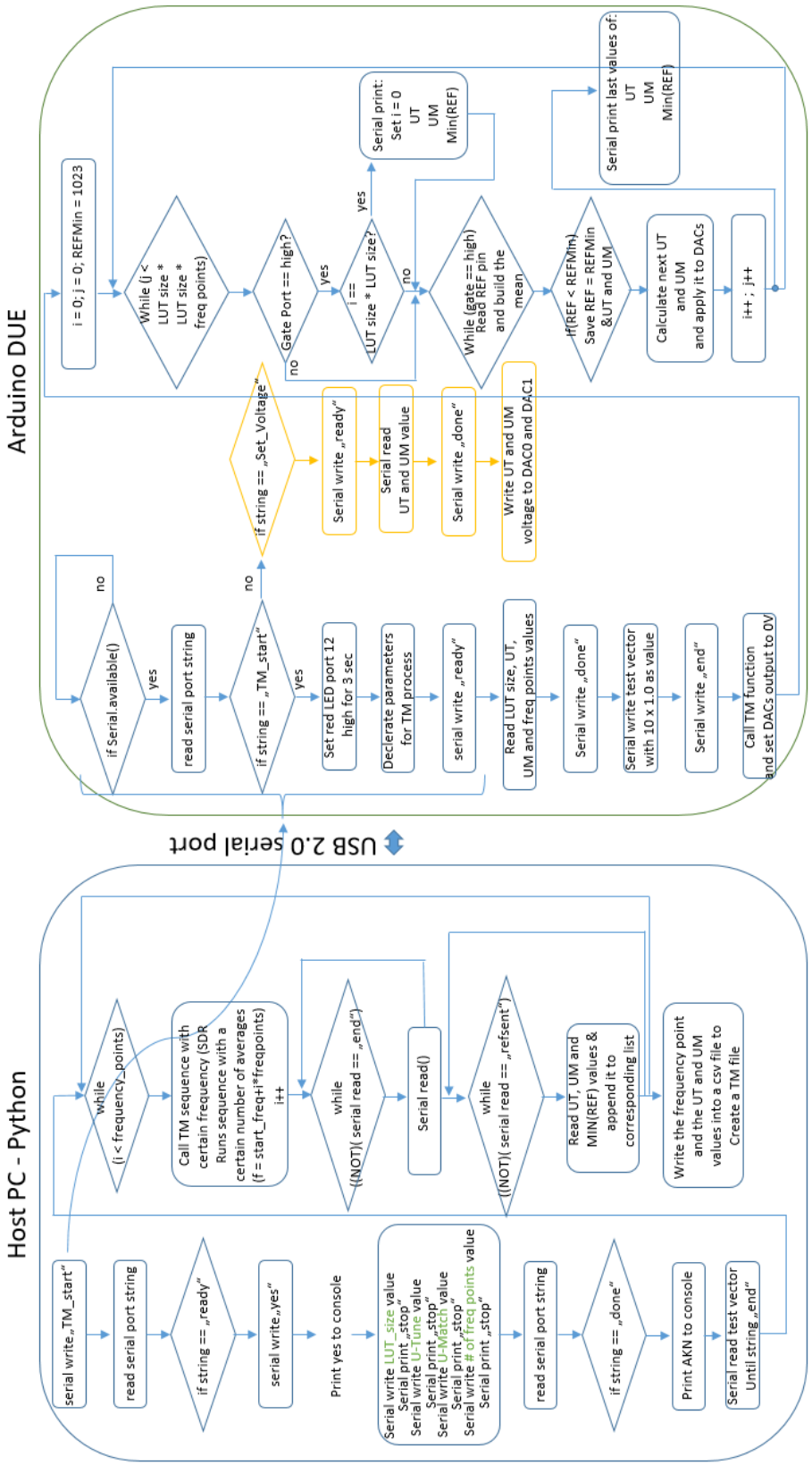

Figure 62: Flowchart of the Tuning- and Matching routine including the host pc part and the 'Arduino Due' part.

To record broadband spectra and also to check the quality of a match, the voltages from the lookup table created by tuning and matching need to be loaded and applied to the probe-head's varactor diodes. The lookup table is stored in a csv file and is read via python. The required tuning and matching voltage is always sent to the 'Arduino Due' before the target frequency is applied using Python. The 'Arduino Due' thus ensures that the RX path is matched during the measurement process.
Figure 63 shows the flow chart for applying the tuning and the matching voltage via the 'Arduino Due'. Here, only the orange printed instructions are needed.

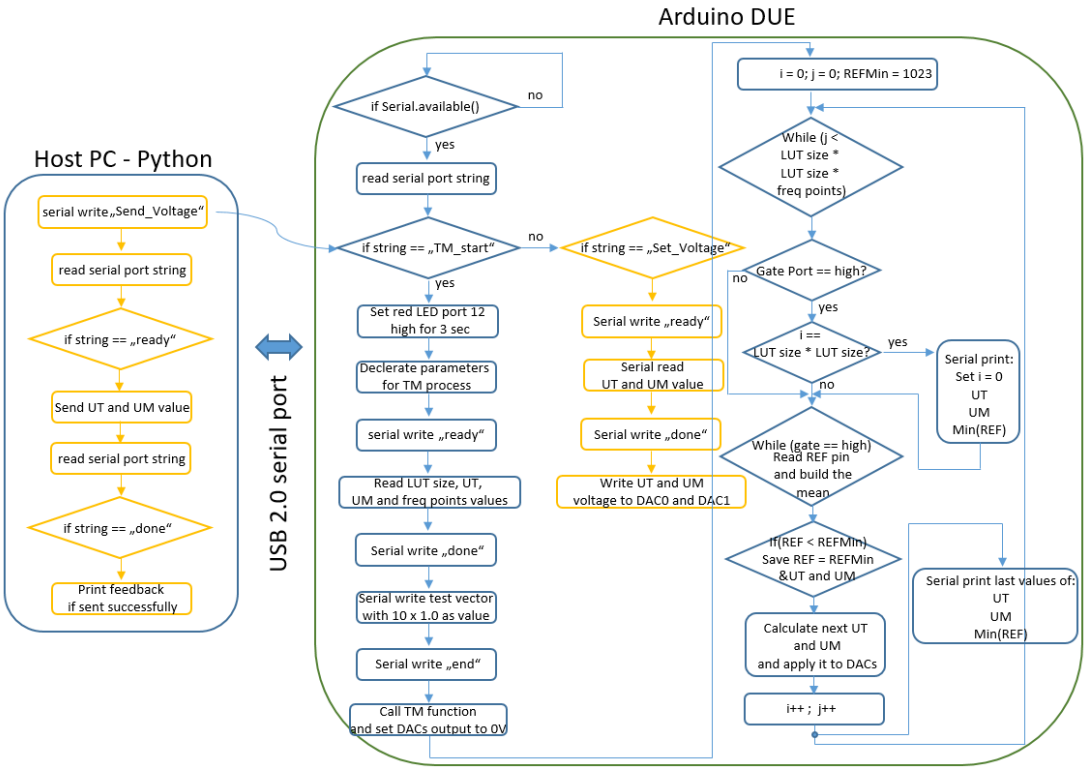

Figure 63: Flow chart for applying the tuning and the matching voltage via the 'Arduino Due'.

The tuning and matching voltages can also be sent manually from Python to the 'Arduino Due'. This is useful to check certain voltage pairs for the quality of the matching or to try voltage combinations manually. To this end the RX port of the probe-head is connected to an NWA and an S_{11} measurement is performed after setting the voltages. The S_{11} plots in figure 106 were generated by connecting the probe-heads RX port to an NWA and by setting the tuning and matching voltages which were determined by the automated routine. Note: when checking the S_{11} for a certain configuration of the tuning and matching voltages, the power from the used NWA must not be bigger then $-20\,dBm$. In this way a lookup table can also be generated manually or values can be recognized and adjusted

where the automatic routine has failed.

2.16.5 Instrumentation for broadband probe-heads

Figure 64 shows the setup used for experiments with broadband probe-heads.

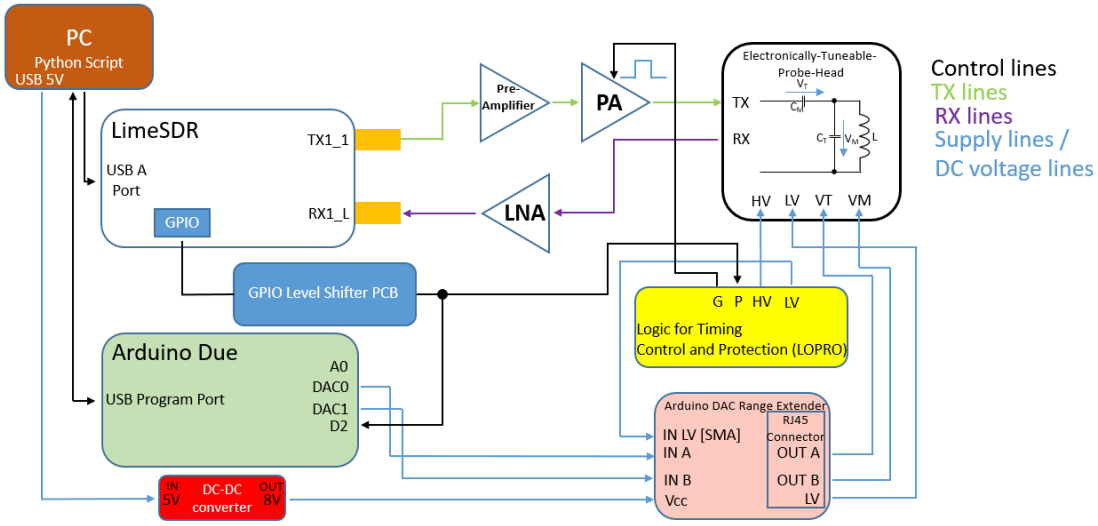

Figure 64: Schematic of the setup for the usage of broadband probe-heads.

Figure 64 above shows all components needed for broadband applications. For recording over several MHz, the probe-head's RX path is tuned and matched using the 'Arduino Due' which applies the voltage pairs stored in the previously created lookup table via the range-extender PCB to the varactor diodes of the probe-head. The TX/RX switching is done by the LOPRO.

2.16.6 Broadband Scans - Measurement Principle

The acquisition principle shall be explained by figure 65. For the measurement, a tuning / matching lookup table must be available over the frequency range to be scanned. Further the frequency step size must be known, so that the number of necessary scans results. It is mandatory to create the tuning and matching file already with the frequency step size to be used. Up to now it is implemented in such a way that the lookup table is loaded and then scan is done over the frequencies in this file.

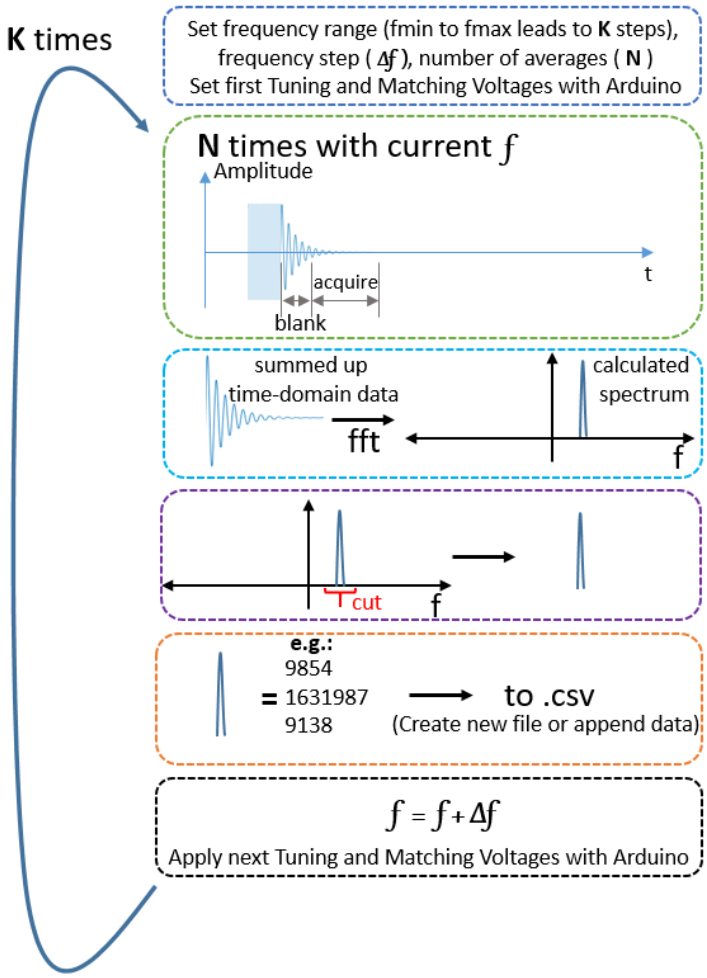

Figure 65: Illustration of the broadband scan strategy.

After the whole acquisition, the data is usually loaded from the generated csv file and plotted.

2.17 Test sequence for broadband scans

For a proof of concept, a broadband scan between $83 - 84\,MHz$ was performed. Figure 66 shows the timing diagram of the FID sequence for broadband scans. The repetition time was set to $T_R = 5\,ms$ and a number of 1000 averages was used. The TX gain of the SDR was set to $50\,dB$. All other settings can be taken from the entries above the last double line of table 5, since they were left the same. The frequency step size used is $100\,kHz$ (so from $83 - 84\,MHz$, there are 11 scans performed).

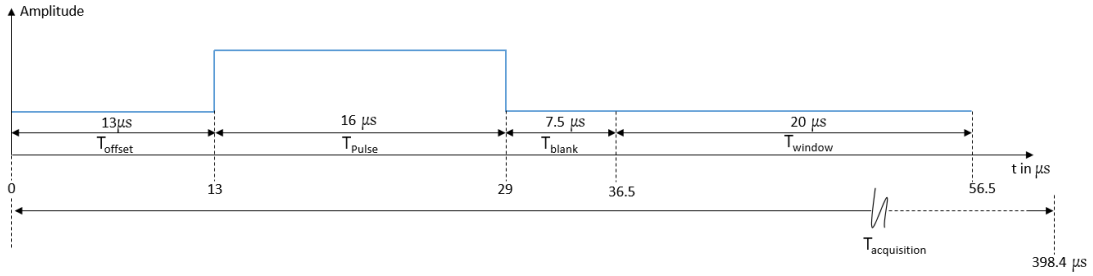

Figure 66: Timing Diagram of the FID sequence for broadband scans. ($T_R = 5\,ms$)

Note: the sequence for the broadband scan was not optimized. It was only done for a proof of concept.

3 Results

3.1 Functionality Check

3.1.1 Time Domain

Figure 67 shows a single pulse and gate signal with a duration of $3\mu s$ and a frequency of $83.5\,MHz$ (with a calibrated TX path).

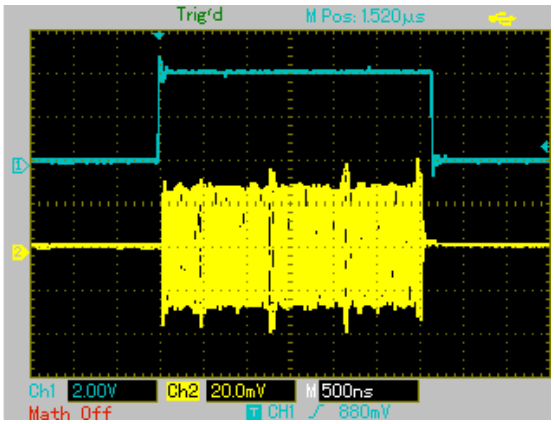

Figure 67: Single Pulse and Gate signal with a duration of $3\mu s$ and a frequency of $83.5\,MHz$ produced by the LimeSDR.

3.1.2 Frequency Domain

Figure 68 shows the spectrum of a continuous wave (CW) test-signal of $83.8\,MHz$ (with a calibrated TX path).

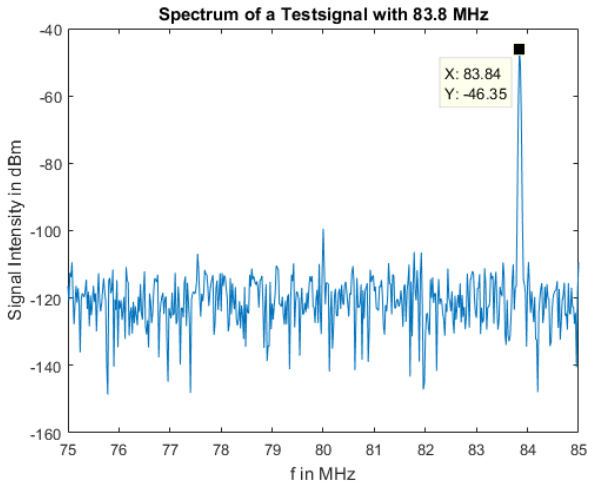

Figure 68: Spectrum of a continuous wave (CW) test-signal of $83.8\,MHz$ produced by the LimeSDR.

Figure 69 shows the spectrum of a pulsed FID sequence with a pulsewidth of $600\,\mu s$ and a repetition time of $1\,ms$ at $83.56\,MHz$.

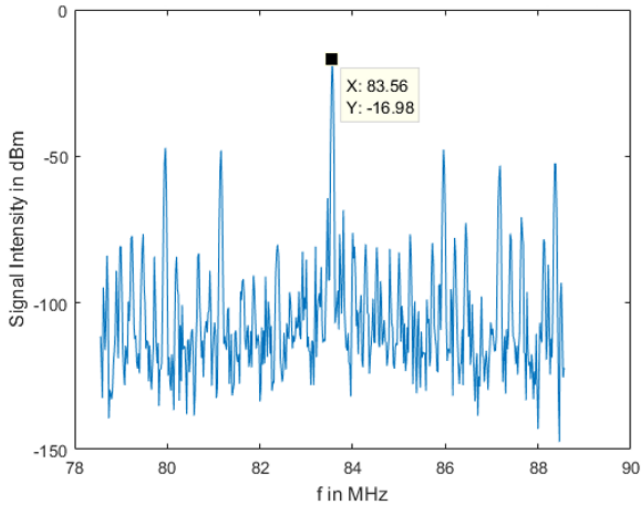

Figure 69: Spectrum of a pulsed FID sequence with a pulsewidth of $600\,\mu s$ and a repetition time of $1\,ms$ at $83.56\,MHz$ produced by the LimeSDR.

3.1.3 TX/RX Loopback

Figure 70 shows an FID sequence looped back to SDRs RX port.

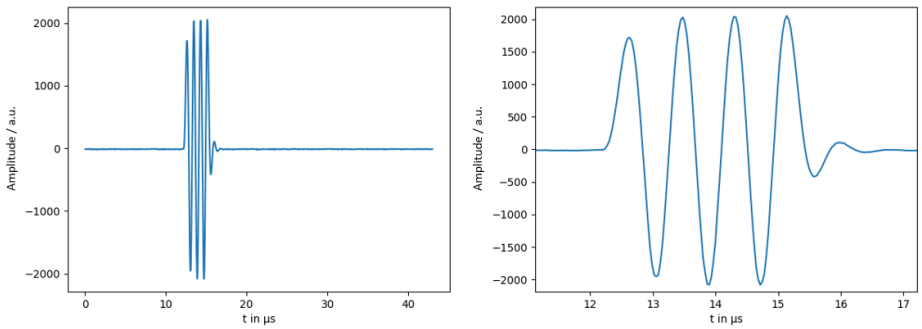

Figure 70: Plot of an FID sequence ($IF = 1.2\,MHz$, target frequency was $83.56\,MHz$) looped back to the SDRs RX port using a $30\,dB$ attenuator between the TX and the RX port. The data was divided by the number of averages which was 1000.

Figure 71 shows a composite pulse sequence looped back to the SDRs RX port.

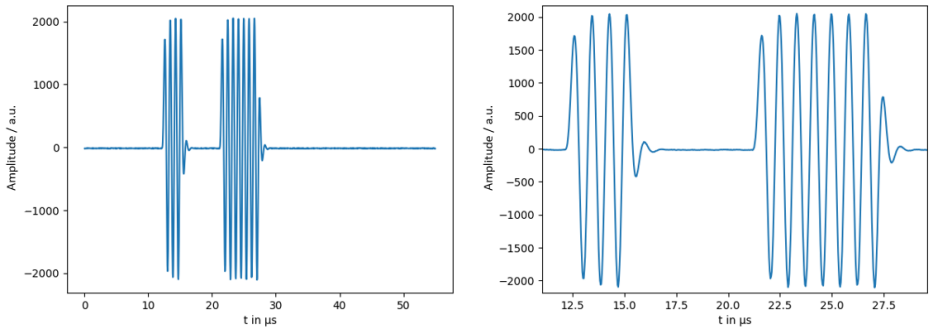

Figure 71: Plot of a Spin-Echo sequence ($IF = 1.2\,MHz$, target frequency was $83.56\,MHz$) looped back to the SDRs RX port using a $30\,dB$ attenuator between the TX and the RX port. The data was divided by the number of averages which was 1000.

Figure 72 shows a phase cycled Spin-Echo sequence looped back to the SDRs RX port. The different traces are the excitation pulses of the Kazan echo (no post-processing) according to table 3.

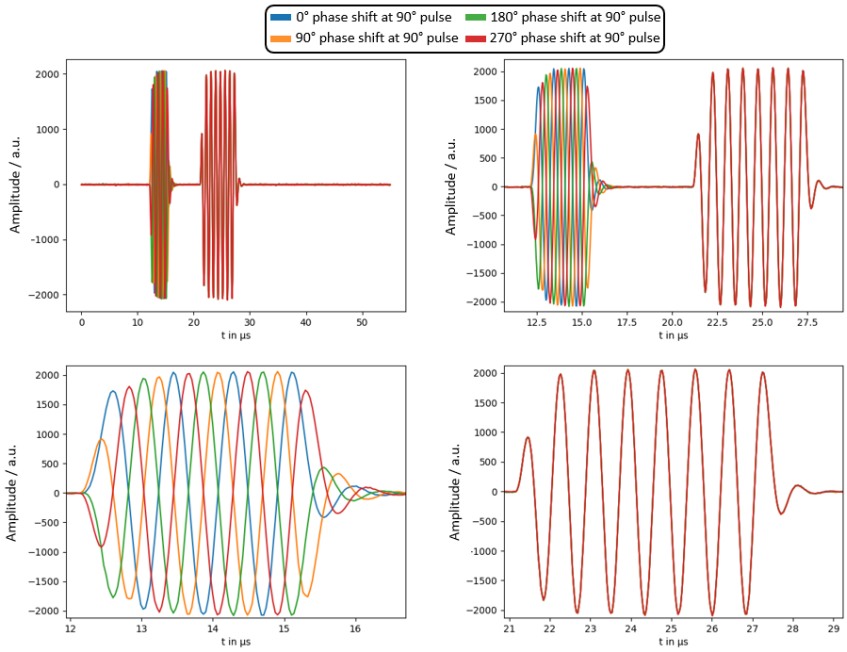

Figure 72: Plot of a phase cycled Spin-Echo sequence ($IF = 1.2\,MHz$, target frequency was $83.56\,MHz$) looped back to the SDRs RX port using a $30\,dB$ attenuator between the TX and the RX port. The data was divided by the number of averages which was 1000.

Figure 73 shows a composite pulse sequence looped back to the SDRs RX port. The different traces are the excitation pulses (no post-processing, only a DC offset correction) of the Composite Pulse according to table 4.

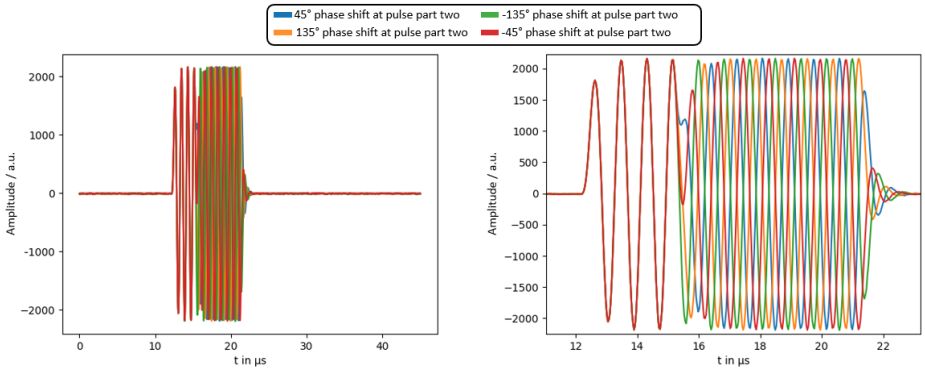

Figure 73: Plot of a composite pulse sequence ($IF = 1.2\,MHz$, target frequency was $83.56\,MHz$) looped back to the SDRs RX port using a $30\,dB$ attenuator between the TX and the RX port. The data was divided by the number of averages which was 1000.

3.2 Characterization of the Assemblies

3.2.1 Monitoring the RF pulse via the directional coupler during experiments

Figure 74 shows a Oscilloscope measurement of an $3\,\mu s$ RF-pulse with a frequency of $83.6\,MHz$ amplified by the PA and measured via the directional couplers forward output.

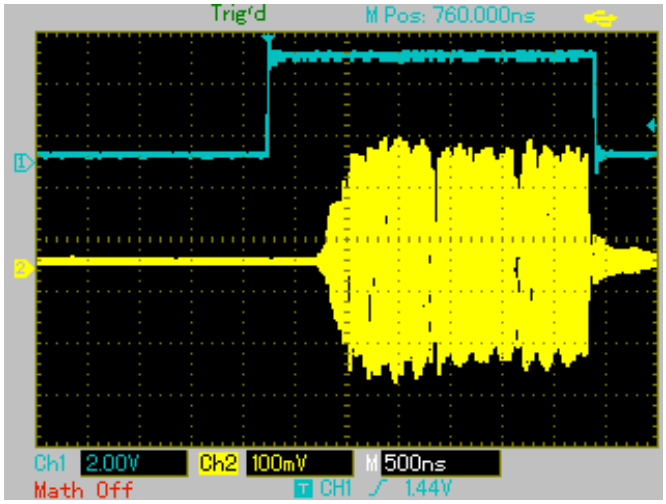

Figure 74: Oscilloscope measurement of an $3\,\mu s$ RF-pulse with a frequency of $83.6\,MHz$ amplified by the PA and measured with the help of the directional couplers forward output.

3.2.2 Pre-Amplifier

Figure 75 shows the S_{11} measurement of the Pre-Amplifier between $100\,kHz$ and $300\,MHz$.

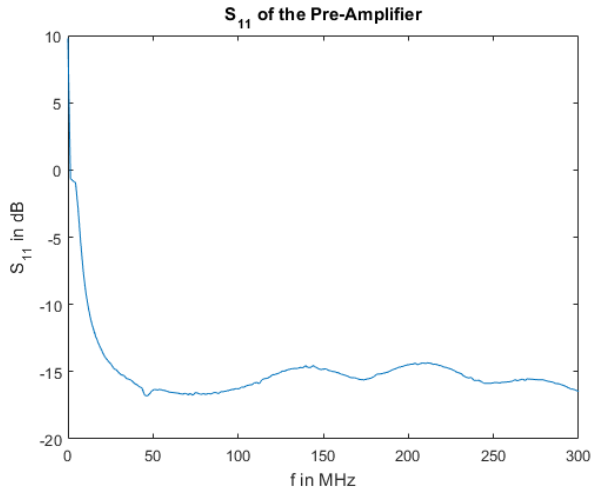

Figure 75: Plot of the S_{11} parameter of the pre-amplifier.

Figure 76 shows the S_{22} measurement of the Pre-Amplifier between $100\,kHz$ and $300\,MHz$.

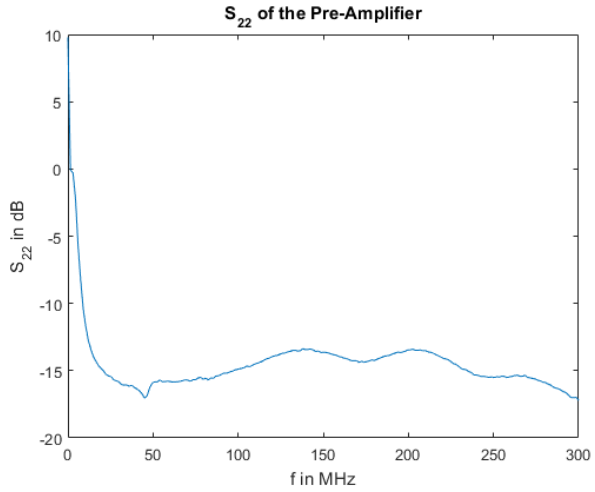

Figure 76: Plot of the S_{22} parameter of the pre-amplifier.

Figure 77 shows the S_{21} parameter of the pre-amplifier in the range from $100\,kHz$ to $300\,MHz$.

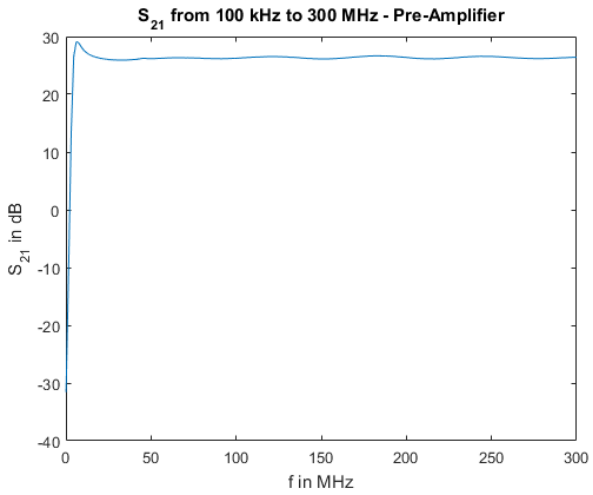

Figure 77: Plot of the S_{21} parameter of the pre-amplifier.

3.2.3 Power-Amplifier

Figure 78 shows the S_{21} measurement of the dummy load and the dummy loads voltage divider from $10\,Hz$ to $200\,MHz$.

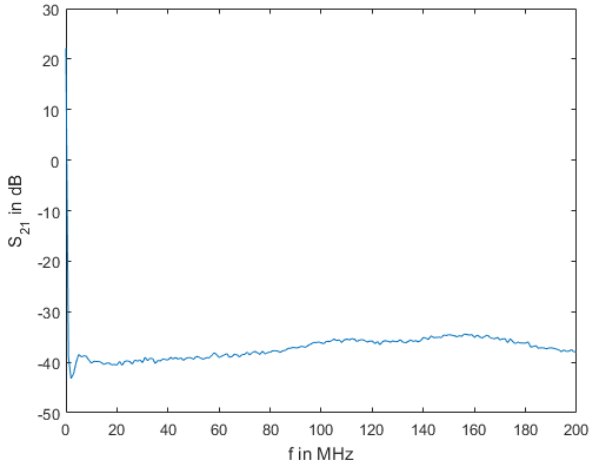

Figure 78: S_{21} of the dummy load from $10\,Hz$ to $200\,MHz$. The division factor at $100\,MHz$ is $-36.15\,dB$.

Figure 79 shows the output voltage of the Power Amplifier measured via the voltage divider of the dummy load. The frequency was 100 MHz and the TX gain was set to 40 dB.

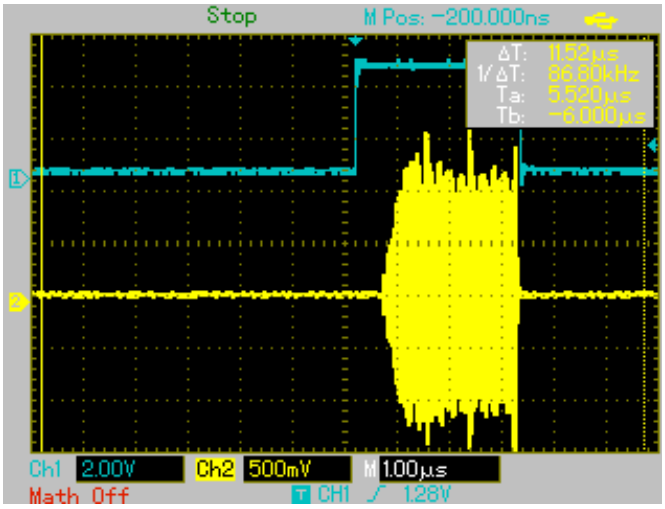

Figure 79: Output voltage of the Power Amplifier measured via the voltage divider of the dummy load.

3.2.4 Sample Coil

Figure 80 shows a S_{11} measurement with a frequency span of 10 MHz sample coil in a matched configuration at 83.56 MHz.

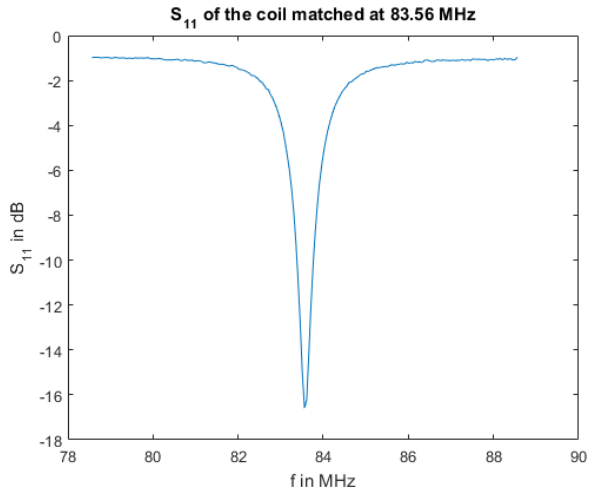

Figure 80: S_{11} measurement of the sample coil matched at 83.56 MHz.

3.2.5 Low Noise Amplifier

Figure 81 shows the S_{11} measurement of the LNA between 100 kHz and 300 MHz.

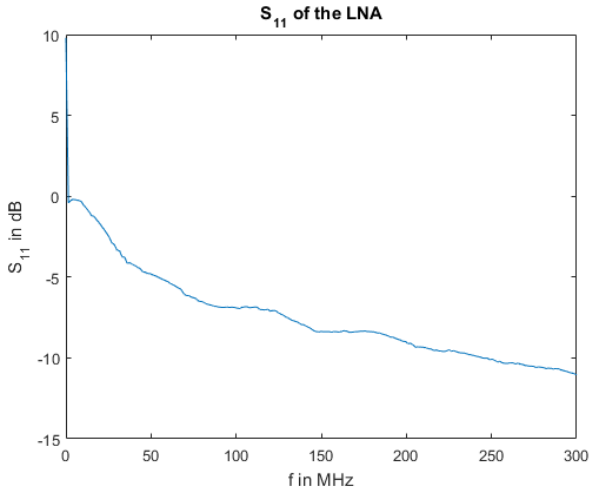

Figure 81: Plot of the S_{11} parameter of the LNA.

Figure 82 shows the S_{22} measurement of the LNA between 100 kHz and 300 MHz.

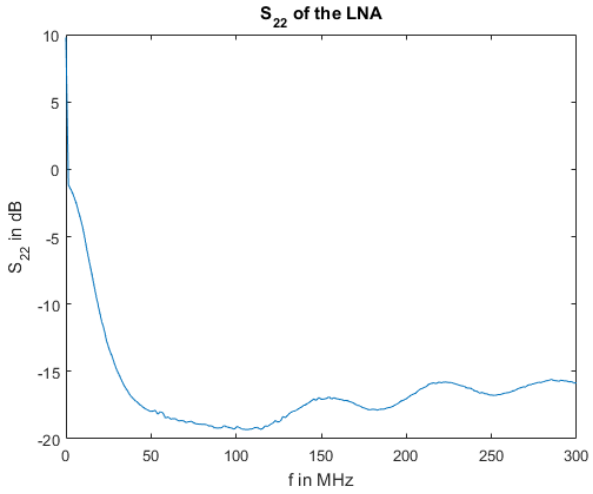

Figure 82: Plot of the S_{22} parameter of the LNA.

Figure 83 shows the S_{21} parameter of the LNA between $100\,kHz$ and $300\,MHz$.

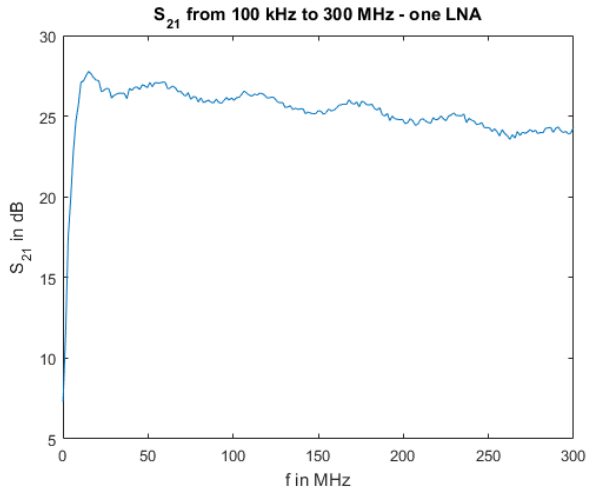

Figure 83: Plot of the S_{21} parameter of the SPF5189Z LNA.

Figure 84 shows the S_{21} parameter of two SPF5189Z in series between $100\,kHz$ and $300\,MHz$.

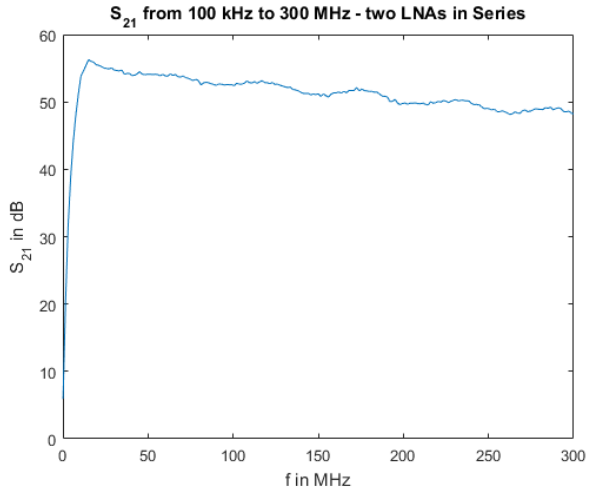

Figure 84: Plot of the S_{21} parameter of a series connection of SPF5189Z.

3.3 Noise Analysis

3.3.1 Conversion Factor

Figure 85 shows the time domain of the TX to RX loopback used for the determination of the scaling factor from a.u. to volts.

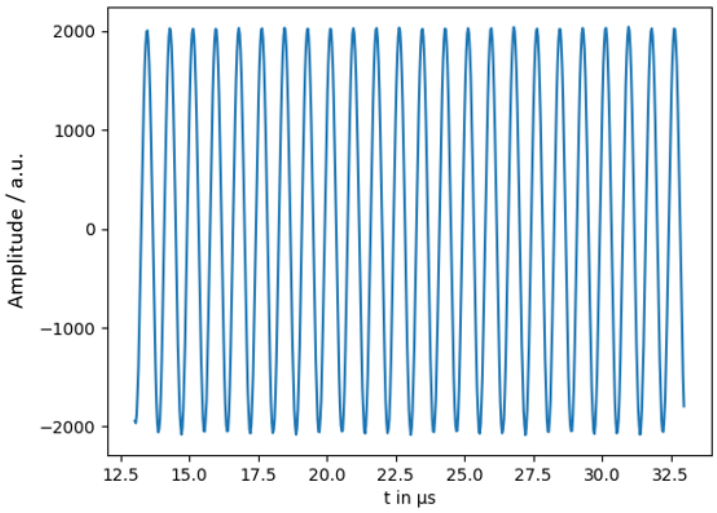

Figure 85: Real part of measured time domain data by loopback of the LimeSDRs TX to RX. The RMS of the real part is 1450.84 a.u..

3.3.2 Measured Noise

Figure 86 shows the noise of a 50 Ω resistor amplified by two SPF5189Z in time domain at a RX bandwidth of 3 MHz.

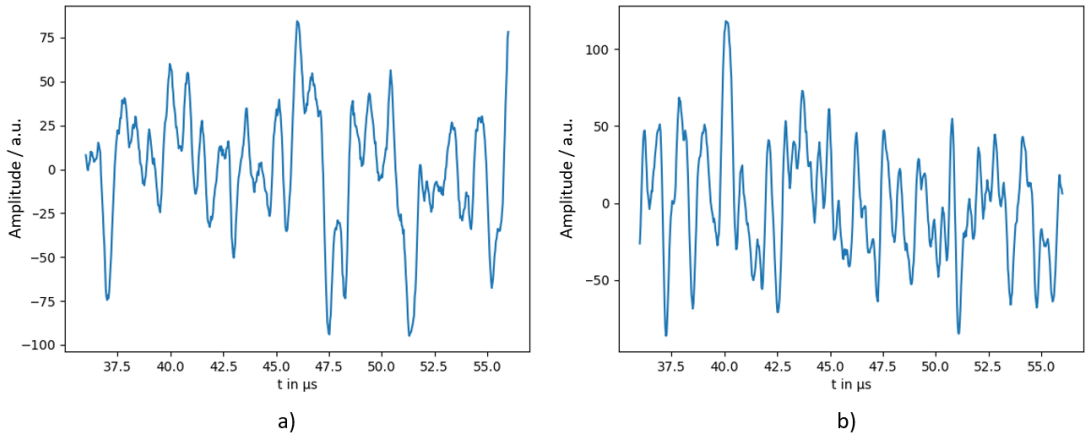

Figure 86: Noise of a) a 50 Ω resistor and b) the matched sample coil amplified by two SPF5189Z in time domain (real part of the signal) at a bandwidth of 3 MHz. The standard deviation of the real part in a) is 33.89 a.u. and in b) 37.78 a.u..

3.4 Reference Measurements between SDR system and SCOUT system on $BiPh_3$

All presented spectra are automatically scaled by python. The plots of signal peaks and noise are therefore not scaled the same, which should be taken care of. Note: all ordinates of the spectra are scaled in a.u. which refer to summed data. This means, no normalization to the number of averages was done.

3.4.1 FID (SDR)

Figure 87 shows the recorded spectra of $BiPh_3$ by using an FID sequence with and without the sample. The spectra are scaled with the arbitrary spectrometer unit at the y-axis.

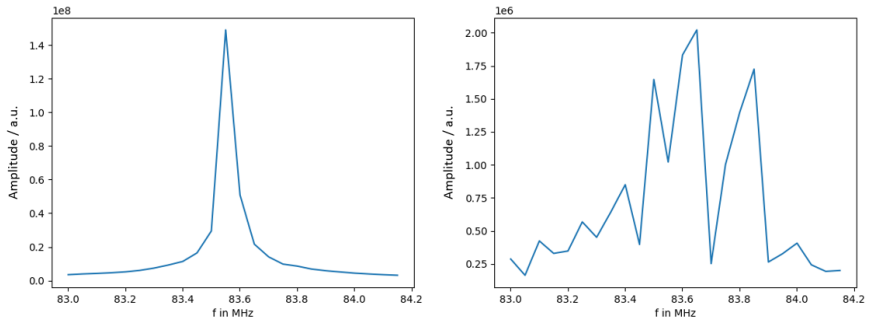

Figure 87: Spectra of $BiPh_3$ using an FID sequence with and without the sample. (left: with sample; right: without sample) Peak: $1.49 \cdot 10^8\, a.u.$; std: $576254\, a.u.$; $SNR = 258.5$.

3.4.2 Spin-Echo (SDR)

Figure 88 shows the recorded spectra of $BiPh_3$ by using a Spin-Echo sequence with and without the sample. The spectra are scaled to represent the signal strength after amplification with the LNA.

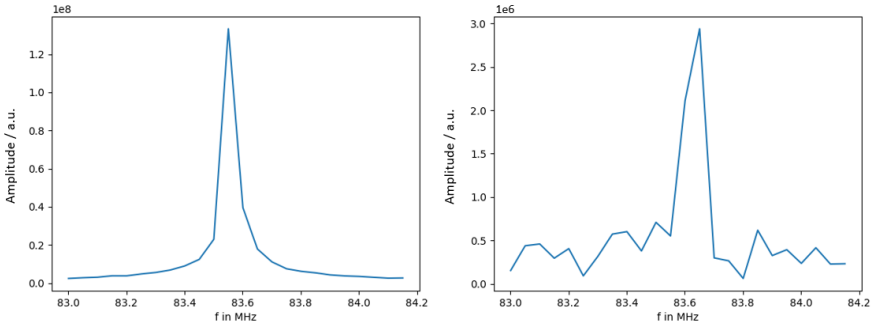

Figure 88: Spectra of $BiPh_3$ using a Spin-Echo sequence with and without the sample. (left: with sample; right: without sample) Peak: $1.333 \cdot 10^8\, a.u.$; std: $629498\, a.u.$; $SNR = 211.76$.

3.4.3 Spin-Echo with Phase-Cycling (SDR)

Figure 89 shows the recorded spectra of $BiPh_3$ by using a Spin-Echo sequence with and without the sample. The spectra are scaled to represent the signal strength after amplification with the LNA.

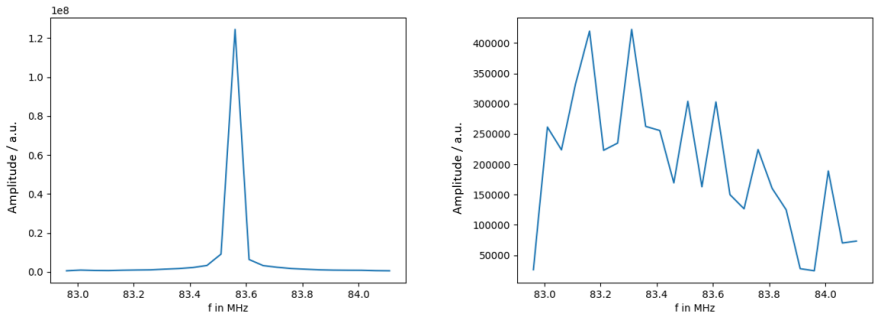

Figure 89: Spectra of $BiPh_3$ using a Spin-Echo sequence with Phase-Cycling with and without the sample. (left: with sample; right: without sample) Peak: $1.244 \cdot 10^8\, a.u.$; std: $110240.9\, a.u.$; $SNR = 1128$.

3.4.4 Composite Pulse (SDR)

Figure 90 shows the recorded spectra of $BiPh_3$ by using a composite pulse with and without the sample.

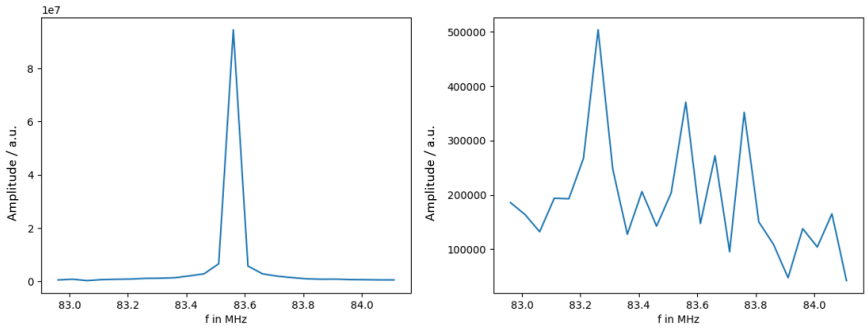

Figure 90: Spectra of $BiPh_3$ using a Composite-Pulse with and without the sample. (left: with sample; right: without sample) Peak: $94484390 \cdot 10^8\, a.u.$; std: $103326.43\, a.u.$; $SNR = 914$.

The following four plots show the measured results with the SCOUT system. The mid frequency of 0.0 MHz stands for the 83.56 MHz of applied frequency, as the SCOUT spectrometer mixes the target RF frequency to 0.0 MHz in the base-band. The y-axis is scaled in the arbitrary unit of the SCOUT spectrometer.

3.4.5 FID (SCOUT)

Figure 91 shows the recorded spectra of $BiPh_3$ by using an FID sequence with and without the sample.

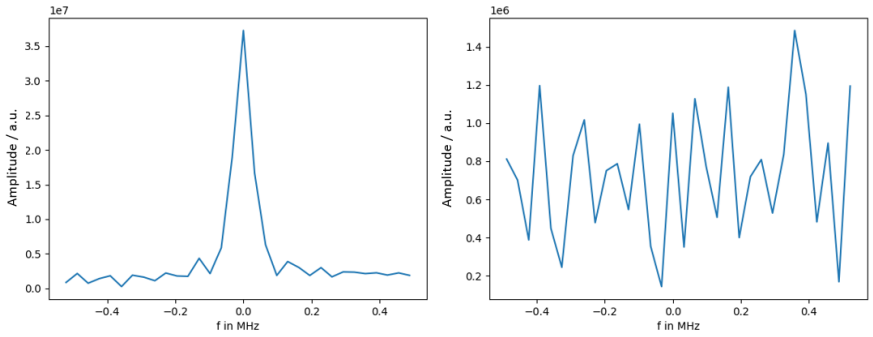

Figure 91: Spectra of $BiPh_3$ using an FID sequence with and without the sample. (left: with sample; right: without sample) Peak: $3.7266 \cdot 10^7$ $a.u.$; std: 335491 $a.u.$; $SNR = 111$.

3.4.6 Spin-Echo (SCOUT)

Figure 92 shows the recorded spectra of $BiPh_3$ by using a Spin-Echo sequence with and without the sample.

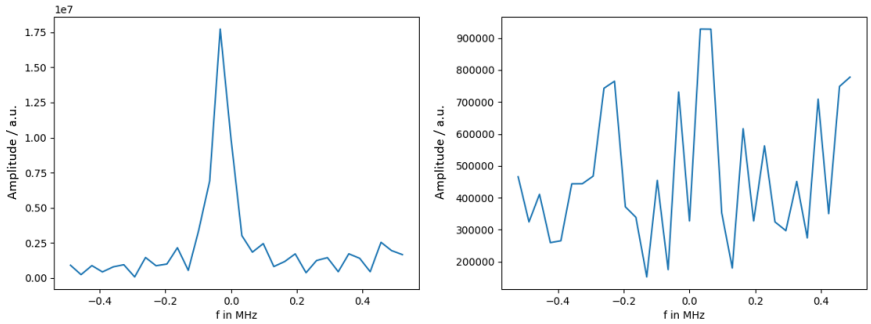

Figure 92: Spectra of $BiPh_3$ using a Spin-Echo sequence with and without the sample. (left: with sample; right: without sample) Peak: $17.72 \cdot 10^6$ $a.u.$; std: 214935 $a.u.$; $SNR = 82$.

3.4.7 Spin-Echo with Phase-Cycling (SCOUT)

Figure 93 shows the recorded spectra of $BiPh_3$ by using a Spin-Echo sequence with Phase-Cycling with and without the sample.

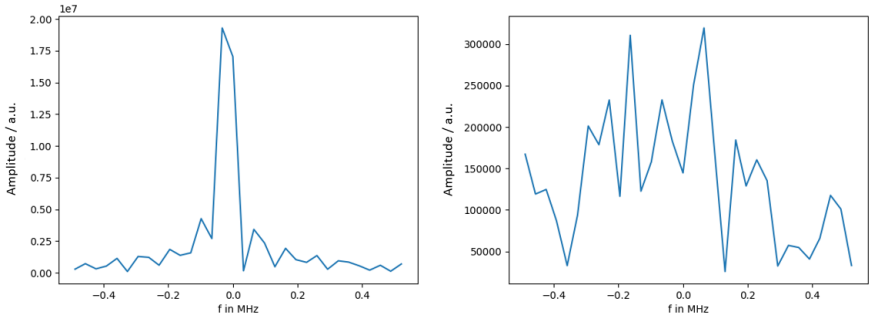

Figure 93: Spectra of $BiPh_3$ using a Spin-Echo sequence with Phase-Cycling with and without the sample. (left: with sample; right: without sample) Peak: $19.3 \cdot 10^6\, a.u.$; std: $76581\, a.u.$; $SNR = 252$.

3.4.8 Composite Pulse (SCOUT)

Figure 94 shows the recorded spectra of $BiPh_3$ by using a Composite-Pulse with and without the sample.

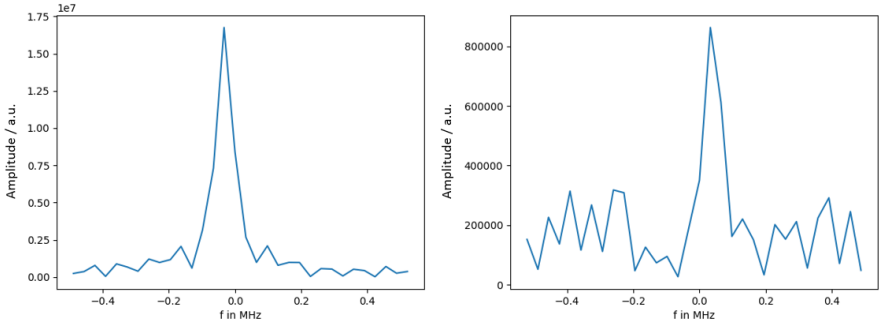

Figure 94: Spectra of $BiPh_3$ using a Composite-Pulse with and without the sample. (left: with sample; right: without sample) Peak: $16.75 \cdot 10^6\, a.u.$; std: $168805\, a.u.$; $SNR = 99$.

3.4.9 SNR Comparison

Table 7 show the SNR comparison between the LimeSDR and the SCOUT system.

Table 7: SNR Comparison between the LimeSDR and the SCOUT system.

SNR comparison between the LimeSDR and the SCOUT system.		
Sequence	SNR LimeSDR	SNR SCOUT
FID	259	111
Spin-Echo	212	82
Spin-Echo with Phase-Cycling	1128	252
Composite-Pulse	914	99

3.5 Artefact Suppression with shorter blanking time on $BiPh_3$

Figure 95 shows a comparison between the different pulse sequences with a blanking time of only $4\,\mu s$ between the end of the RF-pulse and the start of the acquisition window.

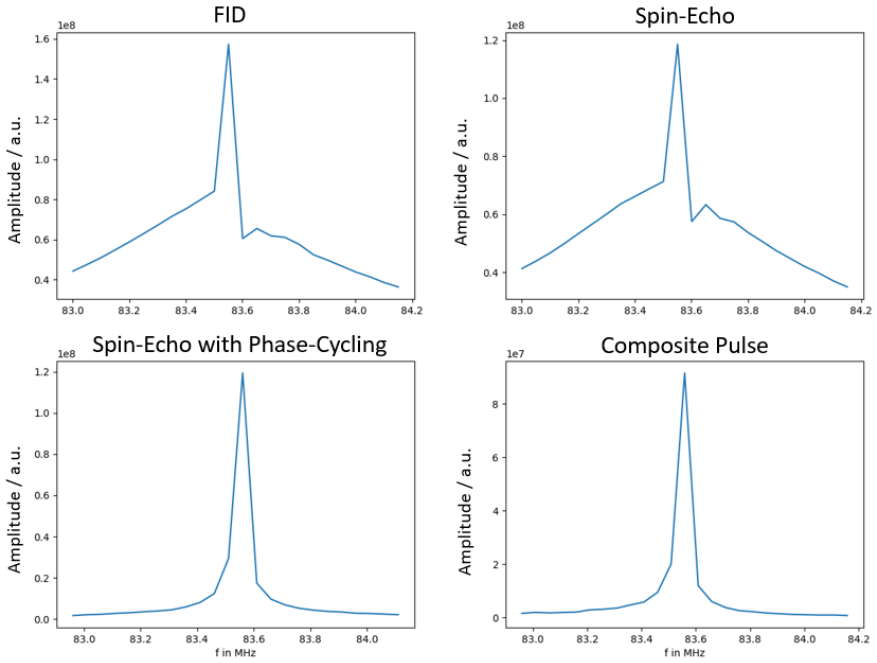

Figure 95: Same sequences as above, but the window (T_{window}) was set closer to the last pulse. For all sequences, the blanking time (T_{blank}) between the last pulse and the begin of T_{window} was set to $4\mu s$. The window length was kept the same ($T_{window} = 20\,\mu s$).

3.6 Further measurements with the SDR spectrometer: Measurements with ground and unground Bi_2O_3

Note: also the following ordinates of the spectra are scaled in a.u. which refer to summed data. This means, no normalization to the number of averages was done.
Figure 96 shows the recorded spectra of unground Bi_2O_3 by using an FID sequence.

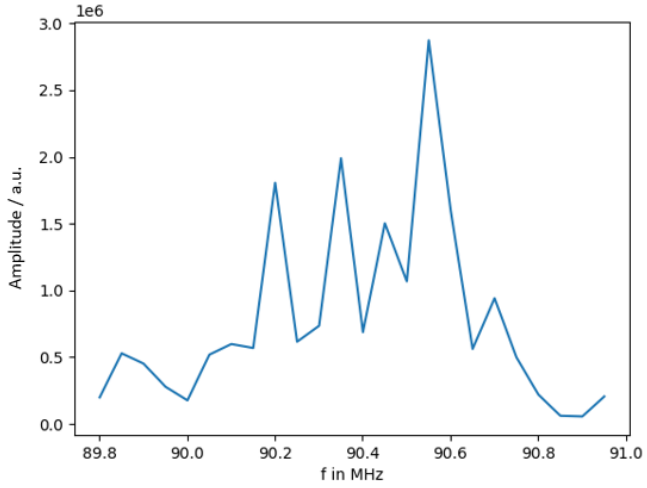

Figure 96: Spectra of unground Bi_2O_3 using an FID sequence with 1000 averages. It was not possible to detect the peak.

Figure 97 shows the recorded spectra of of unground Bi_2O_3 by using a Spin-Echo sequence with and without the sample. The right spectrum (noise) was taken from the reference measurements with $BiPh_3$ (see figure 88) as it was the same sequence and it was captured without a sample (note: therefore the frequency axes are different).

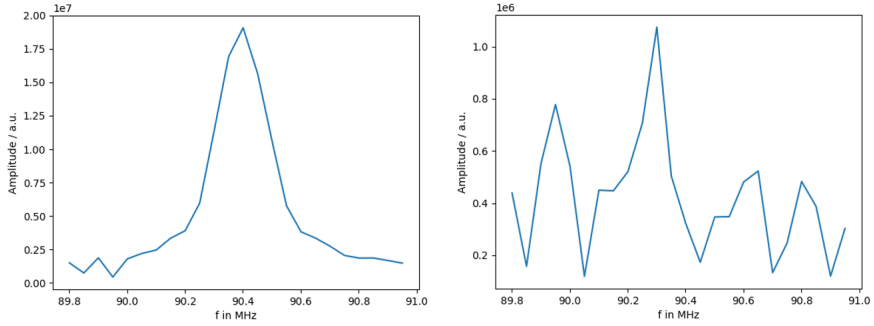

Figure 97: Spectra of unground Bi_2O_3 using a Spin-Echo sequence with and without the sample. 1000 averages were acquired. (left: with sample; right: without sample) Peak: 19070165.51 a.u.; std: 220384.19 a.u.; SNR= 87.

Figure 98 shows the recorded spectra of unground Bi_2O_3 by using a Spin-Echo sequence with phase cycling.

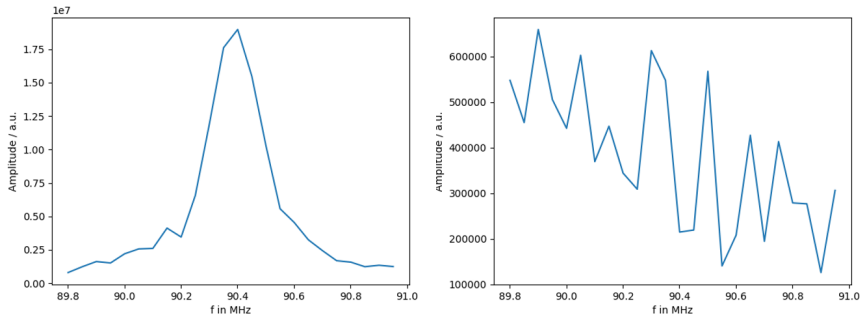

Figure 98: Spectra of unground Bi_2O_3 using a Spin-Echo sequence with phase cycling with and without the sample. 1000 averages were acquired. (left: with sample; right: without sample) Peak: 18975513.83 a.u.; std: 155173.44 a.u.; SNR= 122.

Figure 99 shows the recorded spectrum of unground Bi_2O_3 by using a Composite Pulse.

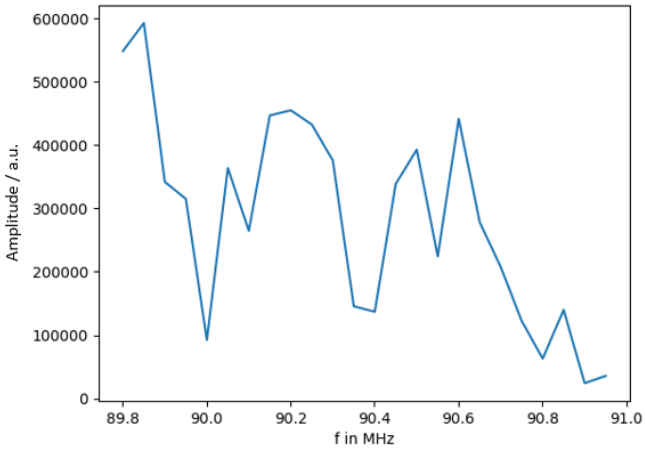

Figure 99: Spectrum of unground Bi_2O_3 using a Composite Pulse. 1000 averages were acquired. It was not possible to detect the peak.

As the SNR for the ground Bi_2O_3 is significantly worse compared to the unground, the averages were set to 10000 for this measurements. This is important when comparing the spectra, because the tenfold number of averages let the peaks appear higher.

Figure 100 shows the recorded spectra of ground Bi_2O_3 by using an FID sequence.

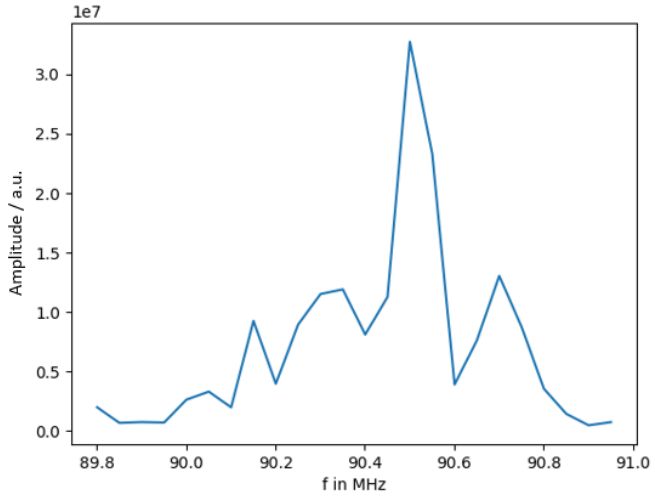

Figure 100: Spectrum of ground Bi_2O_3 using an FID sequence. 10000 averages were acquired. It was not possible to detect the peak.

Figure 101 shows the recorded spectra of ground Bi_2O_3 by using a Spin-Echo sequence. The use of 10000 averages without normalization to the mentioned averages should be pointed out again.

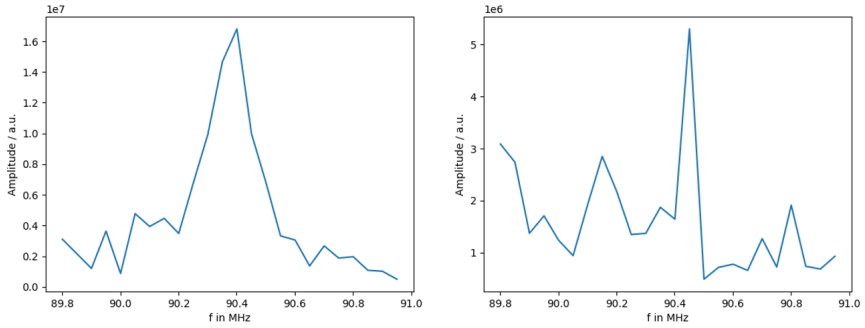

Figure 101: Spectra of ground Bi_2O_3 using a Spin-Echo sequence with and without the sample. 10000 averages were acquired. (left: with sample; right: without sample) Peak: 16805817.02 a.u.; std: 1052954.12 a.u.; SNR= 15.96.

Figure 102 shows the recorded spectra of ground Bi_2O_3 by using a Spin-Echo sequence with phase cycling. The use of 10000 averages without normalization to the mentioned averages should be pointed out again.

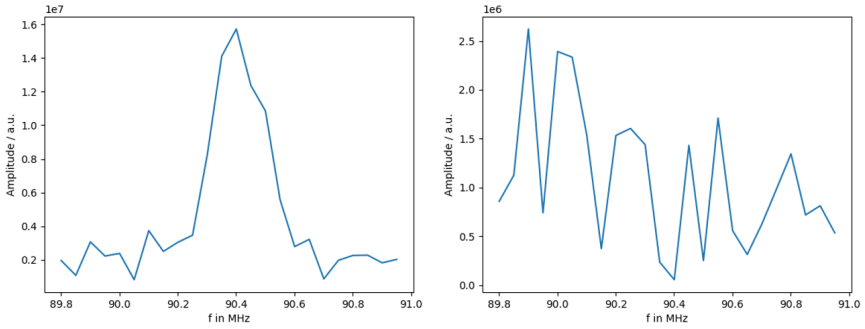

Figure 102: Spectra of ground Bi_2O_3 using a Spin-Echo sequence with phase cycling with and without the sample. 10000 averages were acquired. (left: with sample; right: without sample) Peak: 15731264.35 a.u.; std: 700462, 02 a.u.; SNR= 22.46.

Figure 103 shows the recorded spectra of ground Bi_2O_3 by using a Composite Pulse.

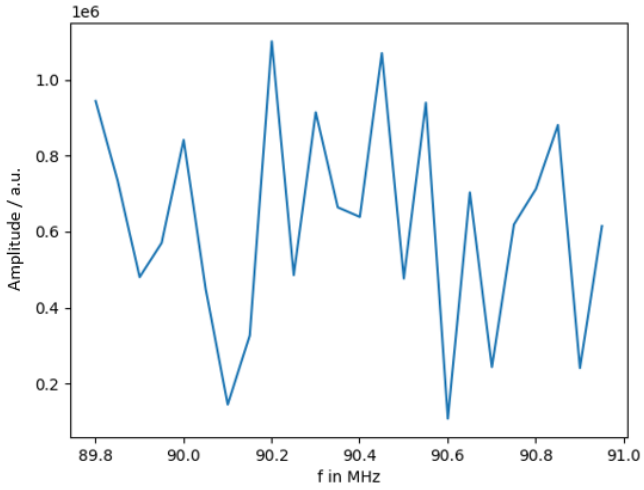

Figure 103: Spectrum of ground Bi_2O_3 using a Composite Pulse. 10000 averages were acquired. It was not possible to detect the peak.

3.7 Broadband Approach

3.7.1 Broadband probe-head matching of TX path

Figure 104 shows the S_{11} measurement of the TX path of the used broadband probe-head.

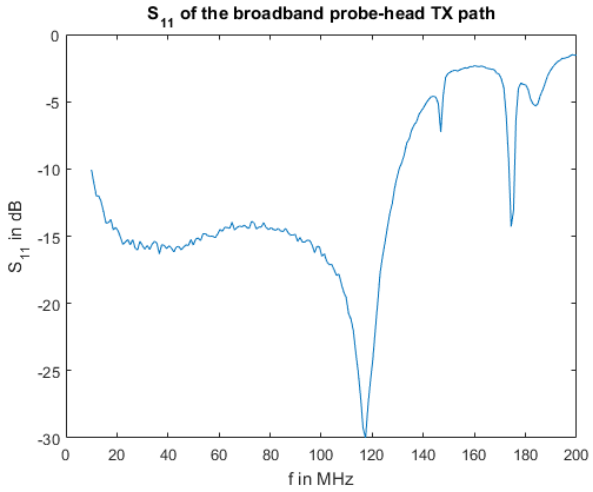

Figure 104: S_{11} of the used broadband probe-heads TX path from a few MHz to 200 MHz.

3.7.2 LOPRO timing

Figure 104 shows the measured timing of the gate of the SDR, gate produced by the LOPRO and the resulting RF-pulse.

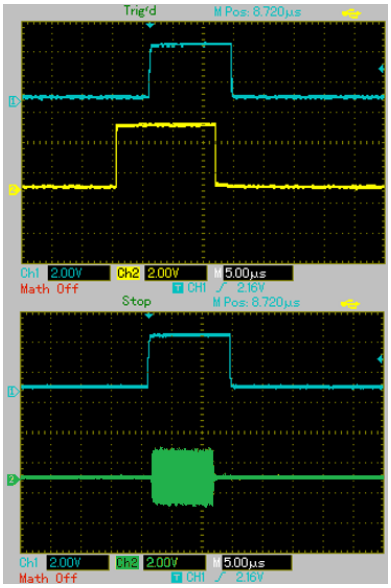

Figure 105: Gate of the SDR (yellow), gate produced by the LOPRO (blue) and the resulting RF-pulse (green).

3.7.3 Automated Tuning and Matching for broadband probe-heads

Figure 106 shows the S_{11} measurement of the RX path of a broadband probe-head (CRYO65-120-GEN-3A) when the minimum was searched by the automated tuning and matching routine.

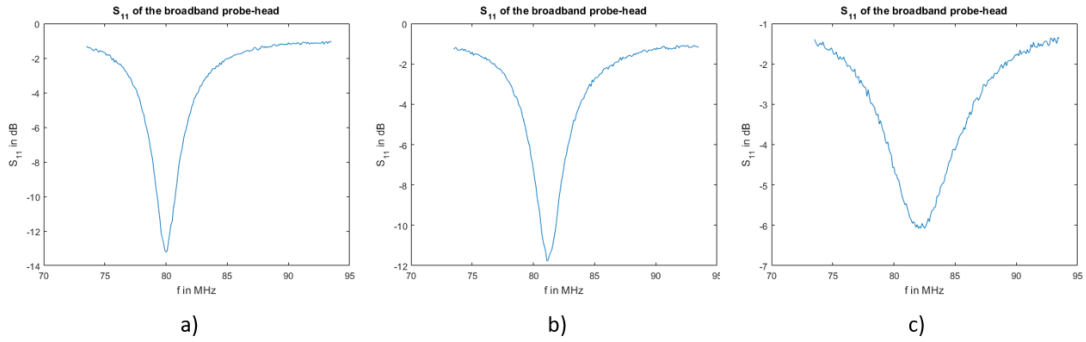

Figure 106: S_{11} measurement of the RX path of a broadband probe-head (CRYO65-120-GEN-3A) when the automated tuning and matching routine was used. The results are aquired for a: 80 MHz, b: 81 MHz and c: 84 MHz.

3.7.4 Broadband Scan

Figure 107 shows a broadband scan of $BiPh_3$ between 83 MHz and 84 MHz with a RX bandwidth of 3 MHz and a number of 1000 averages.

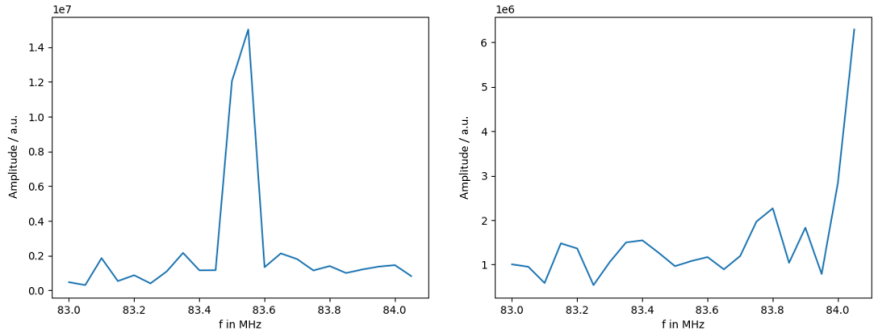

Figure 107: Spectra of $BiPh_3$ using an FID sequence applied to a broadband probe-head with and without the sample. (left: with sample; right: without sample) Peak: $1.5 \cdot 10^7$ $a.u.$; std: 1167492.19 $a.u.$; $SNR = 13$.

3.8 Pictures of the setup for single frequency experiments

Figure 108 shows a picture of the setup.

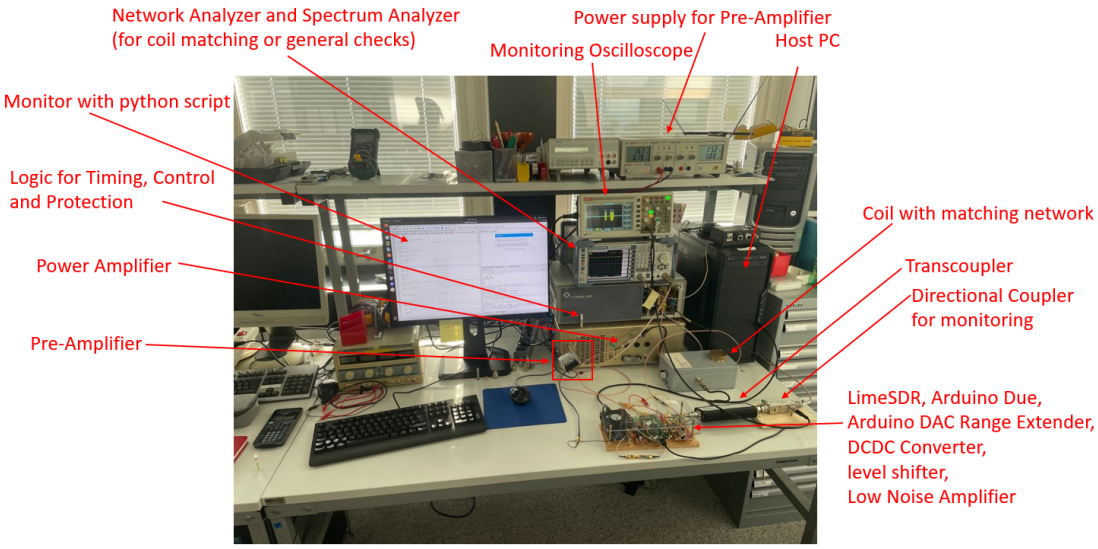

Figure 108: Picture of the setup.

Figure 109 shows a picture of the in this work built console and control unit.

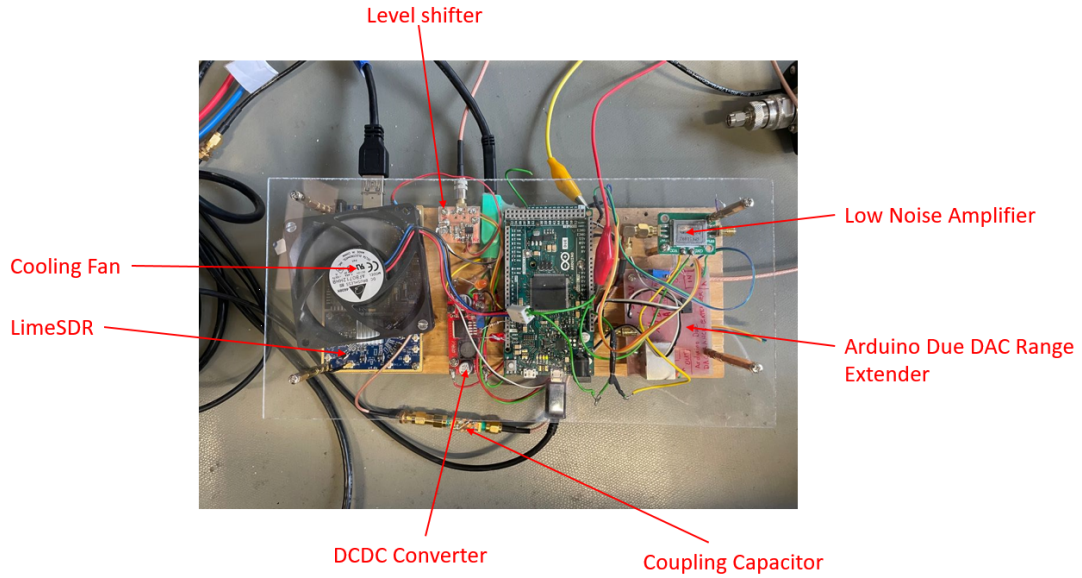

Figure 109: Picture of the console and control unit.

3.9 General Note

Care was taken to always use frequencies between $83 - 84\,MHz$ for the scaling factor evaluation measurements, theoretical noise measurements, and TX tests, since most pulse sequence measurements were made in this range ($BiPh_3$). The reason why there was not always the exact same frequency used is that from time to time slight changes were necessary (e.g. due to heating of the sample because of many measurements in sequence) and that an IF of $3.8\,MHz$ is set per default via the LimeSuiteGUI. That means for the CW test signal which was set with the help of the GUI, the local oscillator frequency was simply set to $80\,MHz$. In this range the TX and RX performance of the LimeSDR remains sufficiently constant.

4 Discussion

4.1 Functionality Check

Looking at figure 67 shows that the gate signal and the RF-pulse produced by the SDR have a similar duration. Figure 68 shows that the TX path is well calibrated, as only a peak at the expected $83.8\,MHz$ can be observed.
However, in figure 69 with pulsed signals, many unwanted frequency components appear. With a tuned resonator (i.e. a matched coil), this is not a problem, since off-resonant components are suppressed efficiently. Note: the pulsed signal was sent to the spectrum analyzer with a $60\,\%$ duty cycle, because with such short pulses as in the application ($\frac{3 \cdot 10^{-6}}{5 \cdot 10^{-3}} = 0.06\,\%$) duty cycle, no signal was measurable.

Figure 70 shows an FID pulse fed back to the SDRs RX port. It can be seen, that the pulse is already down-converted to the base-band frequency of $1.2\,MHz$. The duration of the pulse is about $3\,\mu s$ as expected. The pulse offset of 300 samples leads to a starting point of about $12.5\,\mu s$ and at around $13\,\mu s$ the pulse has settled to the final amplitude. The pulse offset, which can be found in all sequence timing diagrams, was therefore set to $13\,\mu s$ (figures 52, 53 and 55). Figure 71 shows a Spin-Echo sequence determined in the same way. One can clearly see the same offset and that it has the same amplitude compared to the FID.

Figure 72 shows the loopback of a phase cycled Spin-Echo sequence. It can be clearly seen, that the first pulse hast four different phases, whereby the second pulse has four times the same phase. Figure 73 shows the composite pulse, which has the same phase for the first pulse and then four different phases for the second pulse. In all four plots it can be seen that the SDR keeps both the timing and

the amplitude constant.

4.2 Characterization of the Assemblies

4.2.1 Monitoring the RF pulse via the directional coupler during experiments

This measurement demonstrates the difference between the SDR output (figure 67) and the output after the power amplifier (figure 74). In figure 74 it can be observed that the pulse amplified by the PA starts about $500\,ns$ after the gate signal. This means, that the PA has some kind of transient response at the beginning of an applied RF-pulse. This means, that the pulse width is reduced by this $500\,ns$. However, this is a common observation and as long as the pulses have been optimised for the sample, the sequence is optimal.

4.2.2 Pre-Amplifier

The preamplifier reaches between -12 and $-15\,dB$ S_{11} and S_{22} (figure 75 and figure 76) and the gain is very constant at about $26\,dB$ (figure 77) in the investigated frequency range up to $300\,MHz$.
This module worked properly and no problems were observed during operation.

4.2.3 Power-Amplifier

Using the measurement shown in figure 79 and the calculation 42, the propagated power of about $100\,W$ was found to be slightly higher (around $118.6\,W$). Further, the gain was calculated to be about $43\,dB$ according to the calculation 44.
It can be the case, that the PA has a slightly higher output power than expected, but there could also be a slight error due to measurement or reading errors.

4.2.4 Sample Coil

A plot of the S_{11} at $83.56\,MHz$ can be seen in figure 80. The peak value reached was about $-16\,dB$. The amount of reflected voltage in percent for this value lies at $V_{reflected} = 10^{-\frac{16}{20}} \cdot 100 = 15.85\,\%$ and in terms of power one can square this value which yields about $P_{reflected} = 2.51\,\%$. Usually, a S_{11} value lower than $-20\,dB$ is envisaged, but the matching capacitor used reached its lower limit at the target frequency of $83.56\,MHz$. For higher frequencies (e.g., at $90.4\,MHz$ for the Bi_2O_3 measurements), S_{11} values around $-30\,dB$ to $-40\,dB$ can be achieved.

4.2.5 Low Noise Amplifier

The LNA is rather poorly matched at the input, reaching values between only -6 and $-12\,dB$ according to figure 81 S_{11} (between $50 - 300\,MHz$). At the output, according to figure 82, the LNA achieves S_{22} between -15 and $-20\,dB$ (also between $50 - 300\,MHz$). The gain is between 25 and $26\,dB$ in the range of $10 - 300\,MHz$ according to figure 83. The rather poor matching may result because the LNA is built for a broad frequency range (up to $4\,GHz$) and intended to be used at higher frequencies.
This module worked properly and no problems were observed.

4.3 Noise Analysis

The noise analysis was first performed with a $50\,\Omega$ resistor and second, with the matched sample coil. Figure 86 shows the resulting plots and the standard deviations for both cases. As mentioned above, an S_{11} of about $-16\,dB$ was reached (see also figure 80) for the sample coil. As the Noise Analysis showed similar results for the standard deviation (33.89 a.u. for the $50\,\Omega$ resistor and 37.78 a.u. for the coil) - one can assume adequate matching, but the slight deviation might come from both a slight residual mismatch (resistance) and the intrinsic noise figure of the LNA ($0.6\,dB$ at $900\,MHz$, not specified at $83.56\,MHz$). Note: as the noise figure usually increases at higher frequencies, the noise figure at $83.56\,MHz$ should not be bigger than the mentioned $0.6\,dB$, but rather smaller.
The calculated theoretical noise for a $50\,\Omega$ resistor and the bandwidth of $3\,MHz$ was $49.26\,nV$ after 1000 averages while the measurement yielded $52.479\,nV$. In any case, it can be said that for both cases, i.e. regardless of whether the matched coil or the resistor is used as a noise source, values close to the theoretical noise were determined, which means that there is no external interference in the receive path.

4.4 Comparison of the SDR based system with the SCOUT system

The SNR data corresponding to figures 87 - 94 are summarized in table 7. When looking into table 7 , it seems that the SDR based system outperforms the SCOUT spectrometer, no matter which sequence is used. However, it is very important to mention that the two spectrometers were operated at different locations. The SDR system was operated at TU Graz and the SCOUT system at KFU Graz. Both systems were operated outside a shielded chamber in typical laboratory rooms. It was, however, known from previous experiments that the environment

at KFU Graz contains considerably more sources of external interference than at TU Graz. Therefore the measured noise floor was probably higher at KFU. Another finding was that the disturbances at KFU Graz fluctuate strongly (e.g. switching devices). When looking into table 5 and table 6, the settings of the SCOUT system and the SDR based system can be compared. The main differences are, that the SDR based system was set to about twice the sample rate, leading to about double number of points at equivalent acquisition windows (the acquisition windows were tried to set as similar as possible, namely $20\,\mu s$ for the SDR and $19.8\,\mu s$ for the SCOUT system). The sample rate of the SDR based spectrometer was set to $30.72\,MHz$ since most measurements were carried out using this setting and from experience stable operation is guaranteed. In comparison, much smaller dwell times around $0.5 - 1\,\mu s$ (and thus lower sampling rates) were always used in previous measurements with the SCOUT system, since problems always seemed to occur when going lower. In this thesis, however, no problems were observed with a dwell time of $60\,ns$ ($f_s = 16.67\,MHz$), but no experiments were performed with even lower dwell times. A good compromise was sought, where the sampling rates were not too far apart and stable operation was ensured for both devices. When comparing the spectra between SDR and SCOUT system, it can also be seen that the recordings with the SCOUT spectrometer show ripples next to the peak and a smooth line prevails with the SDR. This can be attributed to the fact that the SCOUT system always zero-pads the time data to a power of two (compare Acq. Points and Points 1D in table 6) before it is Fourier transformed. Further, the RX lowpass filter on the SDR system was set to $3\,MHz$ and the RX filter on the SCOUT system was set to $8.33\,MHz$. The gain settings were chosen for both spectrometers to reach about $100\,W$ average power at the output of the PA.

The zero-padding has basically no influence on the SNR, since it only artificially increases the resolution of the spectrum. For an investigation of the influences of the further differences in the settings, a calculation, which was taken from [24], shall be used. This calculation can be found in the appendix and compares the signal of an FID sequence with the expected noise. In this calculation it is shown that the SNR only depends on the acquisition window, when the RX bandwidth is equivalent to the sample rate (with the same number of averages). For the SDR system, as already mentioned, the RX bandwidth was limited by the lowpass filter with a cutoff frequency of $f_c = 3\,MHz$, which would have to be set to $f_c = \frac{f_s}{2} = 15.36\,MHz$ for an exact comparison, as for the SCOUT system the sample rate is equal to the RX bandwidth. Note: the filters are always specified for the one-sided spectrum. This means that when we talk about the RX bandwidth being equal to the sample rate, this is because the filter always

works with $\pm f_c$ (cutoff frequency). For example, the sample rate for the SCOUT system is $f_s = 16.67\,MHz$ and the RX filter is automatically set to $\pm 8.33\,MHz$. In essence, however, this filter offers its advantage only in time domain, since in the spectral domain the bandwidth is given by the acquisition window of $20\,\mu s$, which means $BW_{freqdomain} = \frac{1}{20 \cdot 10^{-6} s} = 50\,kHz$. This means that by choosing sequences that are exactly the same, the systems were actually compared in a fair way, especially by using the same acquisition window.
A minor note should be made here about the PA used. As shown in figure 74, the PA shortens the RF pulse by about $500\,ns$. If the PA used in the SCOUT system does not shorten the pulse, or on the contrary shortens it even more, there could also be some signal degradation due to non-optimal flip angles.
Overall, it must be said that the comparison of the two systems in this case does not show the top performance of each system (especially with the SCOUT, better results could be achieved in a lower noise environment). Thus, no absolute judgement can be made about which system is better. In order to make a better comparison, it would be best to test each system with exactly the same equipment and these tests should be carried out in a shielded chamber (such as is available at the Institute of Biomedical Imaging) in order to shield as many sources of interference as possible.
The reference measurements were intended to check if the performance of the SDR system is similar to that of commercial spectrometers. The data looks promising regarding the performance of the SDR system, which means that also under optimal conditions it could be very close to the performance of the SCOUT spectrometer. What can be said is, that the performance of the SDR system appears to be comparable to the performance of the SCOUT system and thus makes a cost-effective alternative available.

4.5 Further Measurements with the SDR spectrometer

Figure 95 shows the results of all four sequences, when the blanking time (T_{blank}) was reduced to $4\,\mu s$ for all sequences. It can be seen, that the Composite Pulse and the Spin-Echo with phase cycling fulfill their purpose and suppress the ringdown.

Looking on figure 96, it can be seen that it was not possible to measure a signal from the unground Bi_2O_3 using the FID sequence. The same result was obtained with a composite pulse (figure 99). Since it was possible with a Spin-Echo (figure 97) and a Spin-Echo with phase cycling sequence (figure 98), it can be concluded that Bi_2O_3 has too short T_2* to get signals without refocusing pulse (as for an

echo sequence the decay is proportional to T_2). With both Spin-Echos a clear peak at $90.4\,MHz$ can be observed. It should be emphasized that phase cycling improved the SNR by a factor of $\frac{122}{87} = 1.4$. Comparing the SNR value of the phase cycled Spin-Echo with $BiPh_3$, the SNR for Bi_2O_3 is worse by a factor of about 9.25 (since the same sequences and 1000 averages were used, the measurements can be directly compared). These differences show that the system can also measure much weaker samples and that phase cycling has advantages here as well.

By grinding the Bi_2O_3 the distribution of the spins in the powder is much more mixed up. The crystallites obviously suffer from a degradation of their structural order which increases the EFG inhomogeneity considerably. This, in turn, reduces the $T2^*$. Remark: it should also broaden the peak. Therefore, 10000 averages were used for the ground Bi_2O_3.
When looking on figures 100, 101, 102 and 103, it can be seen that the signal peak is again only measureable using a Spin-Echo or a Spin-Echo with phase cycling. As expected, no signal was measured with the sequences FID and composite pulse. When comparing the plots 101 and 102, it is important to consider the different number of averages. The two peaks are identically high, but the ground Bi_2O_3 gives a factor of 10 less signal, because here the summed data are shown and a factor of 10 more averages were used. Due to the grinding, one would also expect a broadening of the peak, which, however, cannot be seen here. Again, phase cycling improved SNR by a factor of $\frac{22.46}{15.96} = 1.4$.
Another finding is that the width of a $BiPh_3$ peak is much narrower than a peak from a Bi_2O_3. This is due to the much shorter T_2^* time of the Bi_2O_3 samples. Comparing the full width half maximum (FWHM) of the two compounds, these are approximately $FWHM_{BiPh_3} \approx 60\,kHz$ and $FWHM_{Bi_2O_3} \approx 200\,kHz$.

4.6 Broadband Application

4.6.1 Broadband probe-heads TX path

As despicted in figure 104, the TX path of the broadband probe-head is matched across a wide frequency range. Here, -13 to $-15\,dB$ are reached from about $20\,MHz$ up to $130\,MHz$.

4.6.2 LOPRO

Figure 60 shows the timing of the SDRs gate signal (yellow), the resulting gate signal produced by the LOPRO (blue) and the RF-pulse (green). It can be seen, that the first delay between the rising edge of the SDRs gate and the rising edge of the LOPROs gate is about $5\,\mu s$. The delay between the falling edges of the

gate signals is about $3\,\mu s$. The SDRs gate signal and pulse length was set to $16\,\mu s$ which is visible when looking onto the yellow colored plot. The resulting pulse length is only about $8\,\mu s$ due to the safety delays.
These delays are not a problem in principle, but they should be considered when optimizing a pulse sequence.

4.6.3 Automated Tuning and Matching

Figure 106 shows three S_{11} traces obtained with automatic tuning and matching. The target frequency was $80\,MHz$ for a), $81\,MHz$ for b) and $84\,MHz$ for c). It can be observed, that the automatic routine leads to a match at the target frequency. However the achieved S_{11} values are only -8 to $-13\,dB$ and thus comparatively poor. In contrast, manual tuning allows for reaching -30 to $-40\,dB$. For adequate matching, at least $-20\,dB$ should be achieved, which unfortunately is not the case. It could be confirmed experimentally that the automated routine works in principle, but needs further improvement.

A general observation is that the width of the S_{11} peak is wider for the broadband probe-heads (see figure 106) than for the manually tunable coil (see figure 80). This indicates that the Q-factor of the resonator is higher for the manually tuneable coil than for the broadband probe-head. Note: the scaling of the frequency axes of the two figures are not the same. The peak of the broadband probe-head looks narrower because the frequency axis covers a larger range.

One possible source of error was found to be the number of selected voltages. Here 50 tuning and 50 matching voltages, i.e. a resolution of $\frac{5V}{50} = 100\,mV$ were used. When entering the voltages manually, it was found that this resolution was sometimes not sufficient to achieve a match below $-20\,dB$. Here, however, runtime is an issue, as at a frequency resolution of $100\,kHz$ and a bandwidth of $10\,MHz$ for 50 x 50 voltages the process already takes around 5 minutes to be finished.
Another source of error is caused by the logarithmic amplifier (AD8310). Figure 110 shows the logarithmic amplifiers output with and without an RF-signal applied. The blue signal is the gate signal of the SDR and the yellow signal is the output of the logarithmic amplifier which receives the reflected wave from the directional coupler.
It can be seen that a zigzag signal with an amplitude of about $100\,mV$ can be measured at the output when no RF signal arrives. Partly this can also be seen between two pulses at the right picture b). If the logarithmic amplifier's output voltage is monitored during the automated routine, the amplitude of the yellow

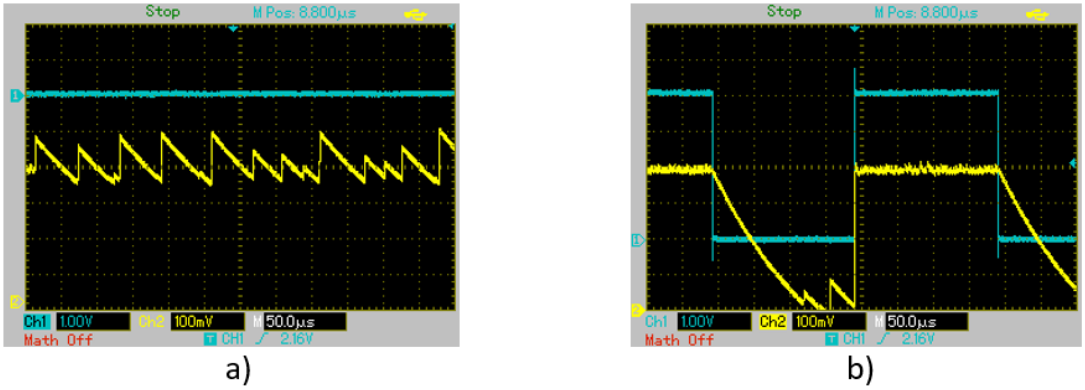

Figure 110: Picture a) shows the output of the logarithmic amplifier when no RF-signal is applied and picture b) shows the output of the logarithmic amplifier with an RF-signal applied.

signal (reflected wave, output of the logarithmic amplifier) moves up and down, depending on the strength of the reflected wave. The minimum of this amplitude is then stored as the optimal match. A closer look shows that there is still some of the zigzag on top of the signal on the amplitude of the yellow signal (also picture b)). The zigzag signal probably prevents the correct detection of the true optima. Further, in picture a), there is a constant offset voltage of around $400\,mV$, even if there is no RF-pulse sent by the SDR. This indicates that the logarithmic amplifier here reacts to interfering signals (for example the zigazag signal).

Spectral components off the target frequency which are present in the TX pulse (figure 69) may contribute significantly to the errors. As mentioned, with already matched coils it does not matter if spurious frequencies appear, because the resonator oscillates only at the resonant frequency. However, when tuning and matching, it can happen that the capacitors match the coil to another than the intended spectral component of the excitation pulse while sweeping through all possible voltage. Thus it can happen that a false minimum is stored.

4.6.4 Broadband Scan

In figure 107 the succesful broadband scan can be seen. The lookup table used here was created manually to ensure a minimum of $-20\,dB$ S_{11} for the RX path at each applied frequency. The frequency step size was $100\,kHz$. The comparatively low SNR ($SNR = 13$) compared to the other measurements is due to the fact that the sequence was not optimized. The entire broadband application was merely a proof of concept, which was confirmed by this experiment. Some improvements and research still need to be done here.

First, the cutting out of the calculated spectrum must be optimized, since here always frequencies which $f >= targetfrequency$ AND $f <= targetfrequency+\Delta f$ was cut out.
The area was intentionally made somewhat larger in order to reliably detect the peak. Here it can be the case that certain frequency points possibly appear twice in the same spectrum (could happen if the cut-out area of one partial measurement overlaps with the next). If there is an overlap of the cut out partial spectra, there are two different effects which falsify the measurement. On the one hand, the signal peak could be measured as too high if two partial spectra constructively overlap exactly at this point. On the other hand, if parts of the noise overlap, it could lead to either higher or lower noise at overlapping points. It would be very important to find a stable approach that is also correct in the frequency domain.

In the methods it was mentioned that the frequency step (Δf) is given by the lookup table. An interpolation step should be added here if frequency points are to be scanned in between.

Comparing figure 66 (the FID sequence for the broadband probe-head) with figure 52 (the FID sequence for the manually matched coil), it can be seen that the broadband probe-head requires longer excitation pulses than the manually matched coil. This is due to the lower Q-factor of the broadband probe-head and thus a lower coil current at the same power.

4.7 Observations with need of further investigations

4.7.1 Unevenly high echo amplitudes with phase-varying sequences

With phase-modulating sequences (i.e. the Composite Pulses and the Spin-Echo with phase cycling) the signal amplitudes differ from those obtained with simple pulses. For a Spin-Echo with phase cycling, for example, one would expect equivalent high RX signals regardless of the phase. Figure 111 shows the Echo amplitudes in time and frequency domain using a Spin-Echo sequence with phase cycling on $BiPh_3$. Figure 112 shows the amplitudes of the different phase shifted RX signals in time domain when using a composite pulse on $BiPh_3$. For both, no post-processing was done (except a DC offset correction). This means, that there was no additional phase shift applied, the signals were plotted as they were received. In order to compare the received signals with the applied pulses, the same legend was used as it was previously for the looped back sequences (figures 72 and 73). The different colored RX traces result from the TX traces with the same color.

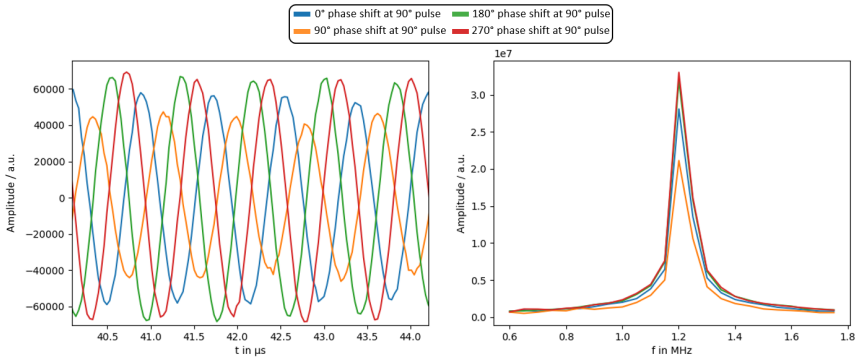

Figure 111: Left: Time domain signals of the Kazan Echo with the respective phase shifts on $BiPh_3$. Right: IF spectrum of the Fourier transformed traces from left (the LO frequency was not added here, the peaks represent the 83.56 MHz peak of $BiPh_3$). Note: the traces are plotted as they were received, which means that no post-processing except a DC offset correction has been done. The signal amplitudes are scaled differently in the time and frequency domain, respectively, and should be ignored for the purposes of this discussion, since the only purpose is to illustrate the different signal magnitudes in each plot.

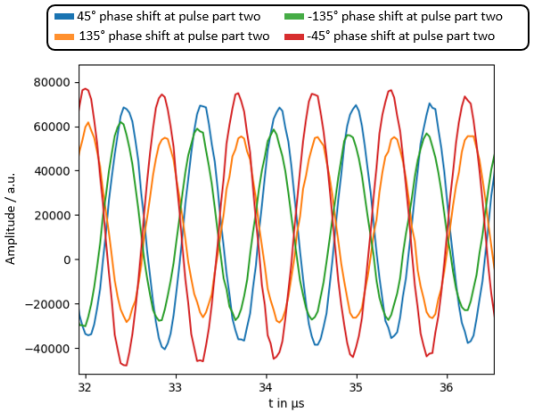

Figure 112: Time domain signals with the respective phase shifts of a Composite Pulse sequence on $BiPh_3$. Note: the traces are plotted as they were received, which means that no post-processing except a DC offset correction has been done.

The reason for this phenomenon is unknown, since a loopback with a similar sequences (figures 72 and 73) does not show any noticeable differences in amplitude. Also the used assemblies like pre-amplifier, power amplifier and LNA cannot be the cause, because they do not contain any phase selective components. The amplitude fluctuations have a certain influence on the quality of the sequence, because with lower partial amplitudes the total signal (created by phase shifting and then adding up the individual traces) is smaller. On the other hand, figure 95 shows that both the composite pulse and the spin echo with phase cycling completely eliminate the ringdown when comparing the spectra of FID and Spin-Echo without phase cycling.

Further measurements and investigations should be performed to verify the reason of the problem. Potential causes are pulse imperfections or purely hardware based problems (for example, that different phase-shifted signals are scaled differently when passing a certain component).

Nevertheless, the experiments clearly prove the efficient suppression of artefacts by the used sequences, even though there are still some minor deficiencies.

4.8 Final note and future outlook

The main goal of this thesis could be clearly reached, i.e. to build the spectrometer based on the available literature and to prove its applicability for NQR spectroscopy. The SDR based concept allows for the setup of a very economic solution which performs comparably with commercial devices. A major advantage of this system built here is that changes and extensions can be made easily.

There remain, however, some issues which should be addressed in the future:
The automatic tuning and matching should be optimized. Here it would be most important to improve the signal quality provided by the logarithmic amplifier. After that, a faster and more reliable search algorithm for the automated tuning and matching should be developed.

The broadband setup works in principle, but different sequences like Spin-Echo, Spin-Echo with phase cycling and Composite Pulse should be tried out and then be optimized. Further, the assembly of the spectrum from the individual measurements should be revised as the current implementation still may contain some minor bugs. However, the foundations for the use of broadband modules were laid and a well-functioning system for single frequency measurements was realized.

Basically all functions which were available with the previous systems ('SCOUT' and 'CONCRADLE') could also experimentally be achieved with the SDR system i.e. single frequency measurements, control and measurement with broadband probe-heads, automated tuning and matching and even phase cycling and Composite Pulses.

4.8.1 Future Outlook

An interesting feature that should definitely be implemented is pulse shaping. One could arrange an arbitrary number of pulses with a resolution of $\frac{1}{sample-rate} = \frac{1}{30.72\,MS/s} = 33\,ns$ in the pulse vectors and choose their amplitudes so that their envelope obeys a certain shape like trapezoid, triangle, Gaussian. This could give advantages in the frequency domain e.g. with respect to the side lobes appearing in the spectrum.

In the future, the SDR-based system could be used for both sensitive measurements as well as for open student labs or lab exercises.

A Appendix

A.1 Setting up the working environment

The 'LimeSDR' can be runned with all common operating systems like Linux, Macintosh and Windows (Note: they are available for purchase at [13]). However, the open-source nature of the SDR invites many users to use Linux. A major advantage of using Linux is that no drivers need to be installed (as long as the 'LimeSuite' which is described in the next section is already installed). Furthermore, the initial project used in this work [2] is based on a Linux system. Here it is easier to get the software functional if a similar operating system is used from the beginning.
It is best to set up an extra Linux partition or a separate hard disk with only Linux. Since the 'LimeSDR' works with very high data rates via USB 3.0 (up to 61.44 MS/s), a native system is also recommended in this regard, since virtual machines often experience data rate drops. In this work, an Ubuntu 20.04 system on a separate hard disk is used.
Furthermore, the basic requirement for the host PC is that it has USB 3.0 ports and Python 3.0 must also be installed on the Ubuntu system.
In general, there is a website `https://wiki.myriadrf.org/Welcome` where a lot of information about the 'LimeSDR' can be found. Many tutorials and general information can be found on this site. There is also a forum where questions can be asked to the community, which are usually answered quite quickly.

A.1.1 Lime Suite GUI

The next step is to install the manufacturer's software. The manufacturer provides a GUI with various features. First of all, you can quickly check whether the board can connect to the PC and whether the TX and RX channels are OK. It can be installed by following the instructions on `https://wiki.myriadrf.org/Installing_Lime_Suite_on_Linux`. Note: point one on this instructions is already sufficient. The shortest and easiest way to install the 'LimeSuite' is using PPA.
To get started, this page `https://wiki.myriadrf.org/Getting_Started_with_LimeSDR-USB_and_LimeSuiteGUI` gives an introduction of how to use the GUI and here `https://wiki.myriadrf.org/LimeSDR-USB_Quick_Test` are some instructions to test the functionality of the SDR.
An important and useful function is the calibration of the IQ mixers, which is explained here `https://wiki.myriadrf.org/Getting_Started_with_LimeSDR-USB_and_LimeSuiteGUI` in section 11.

A.2 Flashing the FPGA gateware to the LimeSDR

First step is to open a new terminal window and to type 'LimeSuiteGUI' into the command line. The modified gateware was taken from [3].

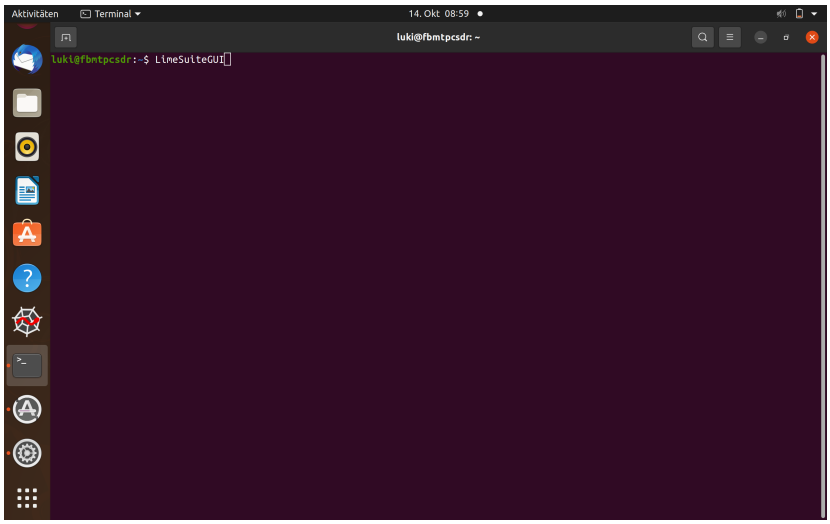

Figure 113: Open LimeSuiteGUI.

Afterwards, a connection to the (already connected via a USB 3.0 cable) LimeSDR board has to be set up. Therefore the menu 'Options' and then the submenu 'Connection Settings' has to be clicked onto.

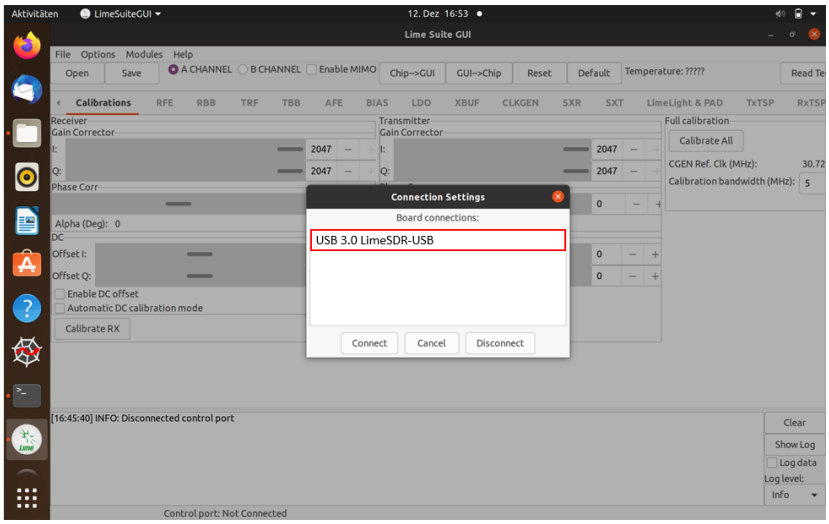

Figure 114: Select the menu to flash the gateware to the FPGA.

For flashing the FPGA Gateware, go to 'Modules' and then select 'Programming' and change the programming mode to 'FPGA FLASH'.

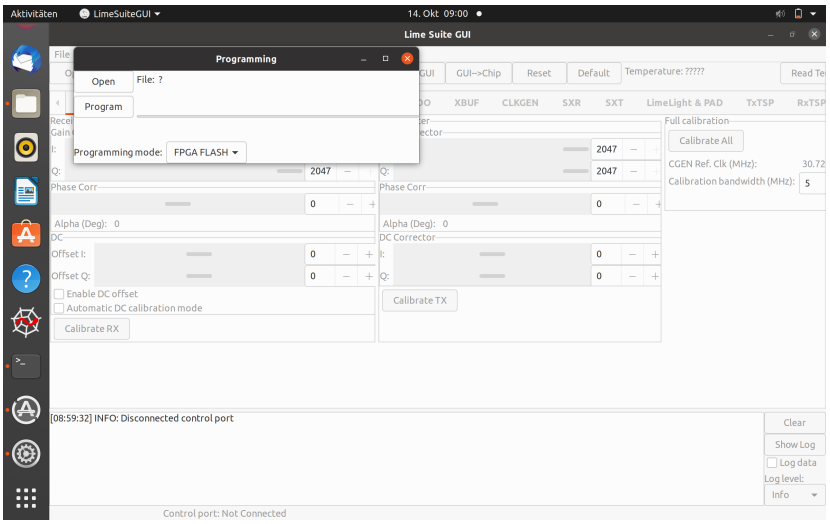

Figure 115: Connect to the LimeSDR board.

Then select the modified gateware file from [3] and press 'Program'.

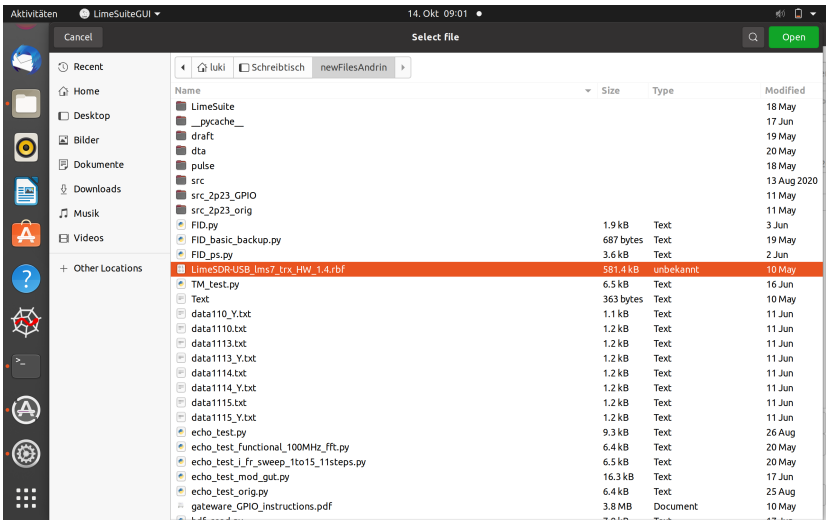

Figure 116: Select the modified FPGA gateware.

A.3 Compile the C++ Routine

Figure 117 shows a screenshot of the first few lines of the C++ routine with the compilation instruction.

```
/**
  @file   N pulses
  @author Andrin Doll
  @brief  N pulses on TX and collection on RX


requirements:
LimeSuite
HDF5 library

compilation:
g++ pulseN_test_USB.cpp -std=c++11 -lLimeSuite -o pulseN_test_USB -I/usr/include/hdf5/serial
 -D_LARGEFILE64_SOURCE -D_LARGEFILE_SOURCE -Wdate-time
 -D_FORTIFY_SOURCE=2 -g -O2 -fstack-protector-strong -Wformat
-Werror=format-security -L/usr/lib/x86_64-linux-gnu/hdf5/serial /usr/lib/x86_64-linux-gnu/hdf5/serial/libhdf5_hl_cpp.a
/usr/lib/x86_64-linux-gnu/hdf5/serial/libhdf5_cpp.a /usr/lib/x86_64-linux-gnu/hdf5/serial/libhdf5_hl.a
/usr/lib/x86_64-linux-gnu/hdf5/serial/libhdf5.a -Wl,-Bsymbolic-functions -Wl,
-z,relro -lpthread -lsz -lz -ldl -lm -Wl,-rpath -Wl,/usr/lib/x86_64-linux-gnu/hdf5/serial


Installation note: this compilation line looks terrifyingly long.. The reason is the inclusion of the HDF5 library,
which is responsible for all the arguments after the -o specification. All this arguments can actually be retrieved from the command:
h5c++ -show
```

Figure 117: Screenshots of the first few lines of the C++ routine. Source: [3].

One has to naviagte into the wished folder and this line has to be executed in a terminal window. The 'limer.py' has to be put into the same folder.

A.4 I/Q Mixer Calibration of the LimeSDRs TX path

First step is again to start 'LimeSuiteGUI' and to connect with the LimeSDR Board. Then, one has to go to select the channel A or B (per default A is selected) depending on which one you want to calibrate. Select the channel and then go to 'SXT' in the menu bar at the top of the GUI. Important: it is recommended to run any of the pulse scripts before doing the settings in the GUI. Trough this all necessary settings are done and the CW signal is output for sure.

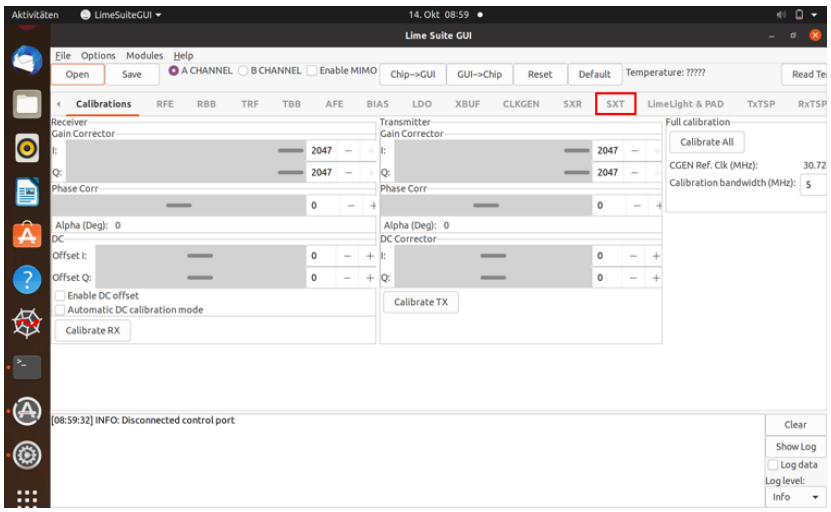

Figure 118: Go to the 'SXT' menu.

Enter the target LO frequency in the 'SXT' menu and then press calculate and tune. Note: the pictures were taken with no connection to the SDR, that's why the error messages appear.

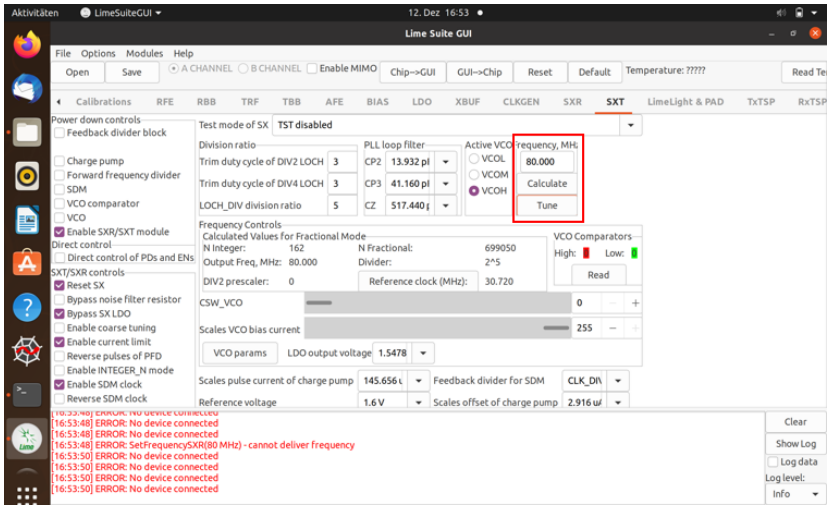

Figure 119: Enter the target LO frequency in the 'SXT' menu. The output RF frequency will be the entered value plus $3.8\,GHz$ of IF frequency which is mixed with the selcted LO frequency.

Then go to the 'TxTSP' menu and select test signal in the red marked area.

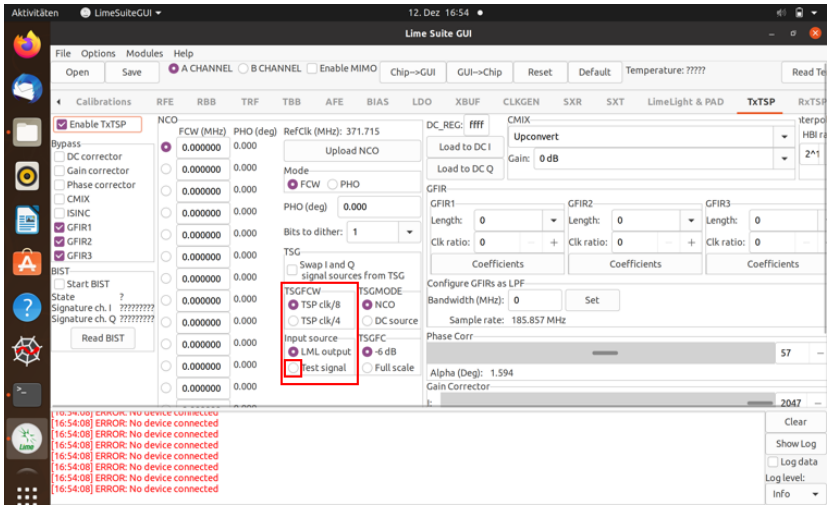

Figure 120: Output the test signal.

Connect the TX of the LimeSDR to a Spectrum Analyzer. The I/Q calibration can then be done using the I/Q DC correction, the I/Q gain correction and the phase correction in the 'TxTSP' menu.

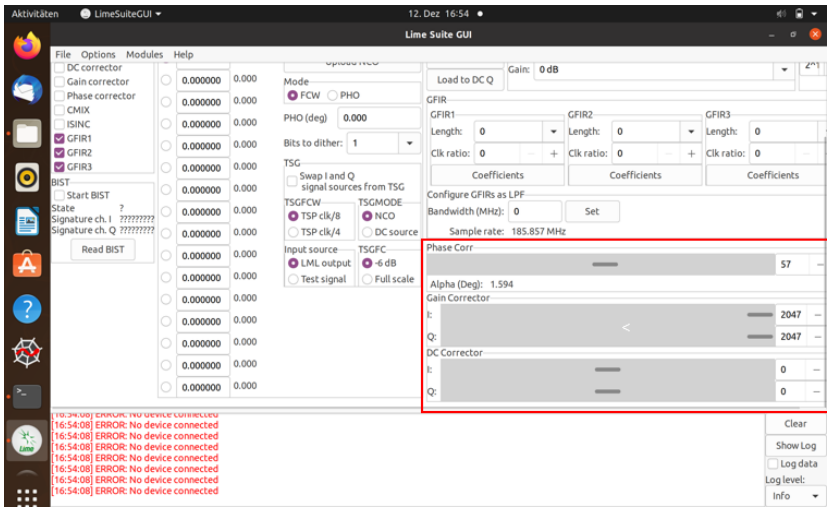

Figure 121: I/Q correction area.

Figure 122 shows a spectrum with a non calibrated TX output. Here, the LO frequency was set to 89.2 MHz and the output of the RF signal is the LO frequency plus the default value for the IF frequency of 3.8 MHz, so 89.2 MHz+3.8 MHz = 93 MHz. The mixers are optimized to 93 MHz as an example and the instructions are only meant to serve as a step by step guide. For the measurements, they were optimized to about 83.8 MHz. Typically, one should not need to make any changes between 83.8 MHz and 93 MHz, as the mixers remain sufficiently well calibrated over this frequency range.

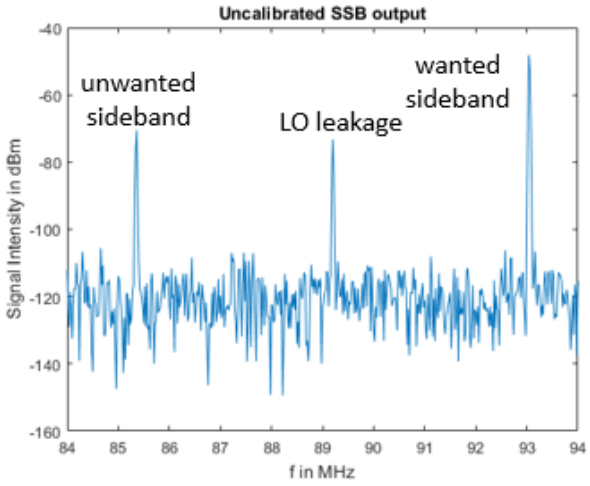

Figure 122: Non calibrated TX channel.

In order to get rid of the LO leakage, the I/Q DC correction bars are used. One can adjust the I and Q DC bars in a way, that the LO disappears from the spectrum (see figure 123). The unwanted sideband can be eliminated using the I and Q gain correction bars and the I/Q phase adjustment bar.

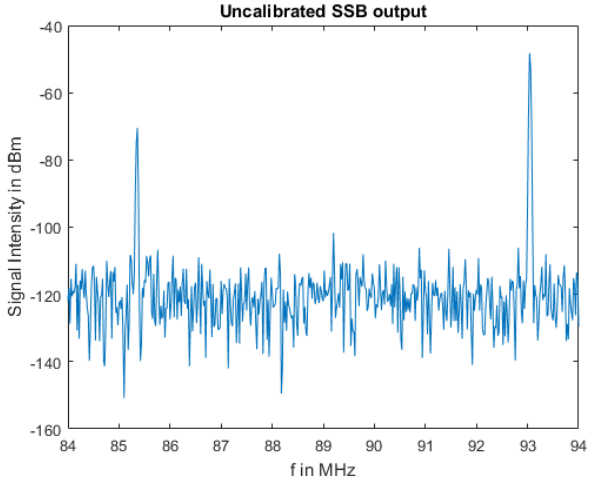

Figure 123: Non calibrated TX channel with corrected LO leakage.

Figure 124 shows a calibrated TX spectrum with only the upper sideband.

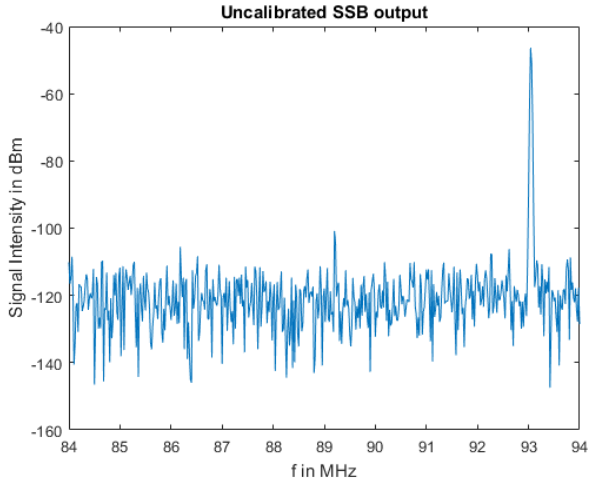

Figure 124: Calibrated TX channel.

A.5 Screenshots of the python scripts for the sequences or other evaluations presented in this work

All sequences were created using the provided code in [2] and [3] as a basis. Further, the author of [2] provided further test sequences on request.

A.5.1 FID

Figure 125 shows the pulse generating part of the used FID sequence.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq
import limr

l = limr.limr('./pulseN_test_USB.cpp');

l.noi = -1                                  # hardcoded initialization of the lime. needed if parameters
                                            # (e.g. Gain, tgi, tdi, are changed and need to be set to chip)

tgtfreq = 83.56e6                           # target frequency of the experiment
if_frq = 1.2e6                              # IF or base band frequency

l.lof = tgtfreq-if_frq                      # LO frequency (target frequency - base band frequency)
l.sra = 30.72e6                             # Sampling Rate
l.nav = 1000                                # number of averages
l.nrp = 1                                   # number of repetitions

l.tdi = -45                                 # TX I DC correction
l.tdq = 0                                   # TX Q DC correction
l.tgi = 2047                                # TX I Gain correction
l.tgq = 2039                                # TX Q Gain correction
l.tpc = 3                                   # TX phase adjustment

l.rgi = 2047                                # RX I Gain correction
l.rgq = 2047                                # RX Q Gain correction
l.rdi = 0                                   # RX I DC correction
l.rdq = 0                                   # RX Q DC correction
l.rpc = 0                                   # RX phase adjustment


l.trp = 5e-3                                # repetition time
l.tac = 82e-6                               # acquisition time (gives minimum buffer size)
l.t3d = [1, 0, 50, 10]                      # GPIO Pin3 is centered around the pulse

l.pfr = [if_frq]                            # pulse frequency
l.pdr = [3e-6]                              # pulse  duration
l.pam = [1]                                 # relative pulse amplitude
l.pof = [300]                               # pulse arrangement in samples [300] -> 12.5 us offset


l.npu = len(l.pfr)                          # number of pulses

l.rgn = 55.0                                # RX gain
l.tgn = 40.0                                # TX gain


l.rlp = 3.0e6                               # RX lowpass
l.tlp = 130.0e6                             # TX lowpass

l.spt =  './pulse/FID'                      # directory to save to
l.fpa = 'setup'

l.run()
```

Figure 125: Screenshot of the used FID sequence (pulse generating part).

Figure 126 shows the signal processing part of the used FID sequence.

```
#reads back the file which was recently saved
l.readHDF()

#evaluation range, defines: blanking time and window length
evran = [23, 43]
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

#time domain x and y data
tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

#correcting a offset in the time domain by subtracting the mean
tdy_mean = tdy-np.mean(tdy)

#fft of the corrected time domain data
fdyl = fftshift(fft(tdy_mean,axis=0),axes=0)

#fft freq and fft shift is here used to scale the x axis (frequency axis)
fdxl = fftfreq(len(fdyl))*30.72
fdxl = fftshift(fdxl)

plt.figure(1);
plt.plot(tdx,tdy_mean/l.nav)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

#add LO frequency
lof=l.HDF.attr_by_key('lof')
for i in range(0, len(fdxl)):
    fdxl[i] = fdxl[i]+lof[0]/1e6


#selecting the interessting part of the spectrum
shifter = 12
stopper = 270
#here the right side of the spectrum is selected
y=abs((fdyl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]))
x=fdxl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]

plt.figure(5);
plt.plot(x, y)
plt.xlabel("f in MHz")
plt.ylabel("Amplitude / a.u.")
plt.show()

#print std (for SNR determination -> noise analysis without sample)
print("std rms frequency domain next to peak X: " + str(np.std(y)))
#print max of fft (for SNR evaluation - should give peak maximum)
print("MAX of Signal: " + str(max(y)))
```

Figure 126: Screenshot of the used FID sequence (signal processing part).

A.5.2 Spin-Echo

Figure 127 shows the pulse generating part of the used Spin-Echo sequence.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq
import limr

l = limr.limr('./pulseN_test_USB.cpp');

l.noi = -1                                          # hardcoded initialization of the lime. needed if parameters
                                                    # (e.g. Gain, tgi, tdi, are changed and need to be set to chip)

tgtfreq = 83.56e6                                   # target frequency of the experiment
if_frq = 1.2e6                                      # IF or base band frequency

l.lof = tgtfreq-if_frq                              # LO frequency (target frequency - base band frequency)
l.sra = 30.72e6                                     # Sampling Rate
l.nav = 1000                                        # number of averages
l.nrp = 1                                           # number of repetitions

l.tdi = -45                                         # TX I DC correction
l.tdq = 0                                           # TX Q DC correction
l.tgi = 2047                                        # TX I Gain correction
l.tgq = 2039                                        # TX Q Gain correction
l.tpc = 3                                           # TX phase adjustment

l.rgi = 2047                                        # RX I Gain correction
l.rgq = 2047                                        # RX Q Gain correction
l.rdi = 0                                           # RX I DC correction
l.rdq = 0                                           # RX Q DC correction
l.rpc = 0                                           # RX phase adjustment


l.trp = 5e-3                                        # repetition time
l.tac = 82e-6                                       # acquisition time (gives minimum buffer size)
l.t3d = [1, 0, 50, 10]                              # GPIO Pin3 is centered around the pulse

# pulse durations
l.pfr = [if_frq, if_frq]                            # pulse frequency
l.pdr = [3e-6, 6e-6]                                # pulse duration
l.pam = [1, 1]                                      # relative pulse amplitude
l.pof = [300, np.ceil(9e-6*l.sra)]                  # pulse arrangement in samples [300] -> 12.5 us offset

l.npu = len(l.pfr)                                  # number of pulses

l.rgn = 55.0                                        # RX gain
l.tgn = 40.0                                        # TX gain


l.rlp = 3.0e6                                       # RX lowpass
l.tlp = 130.0e6                                     # TX lowpass

l.spt =  './pulse/ECHO'                             # directory to save to
l.fpa = 'setup'

l.run()
```

Figure 127: Screenshot of the used Spin-Echo sequence (pulse generating part).

Figure 128 shows the signal processing part of the used Spin-Echo sequence.

```
#reads back the file which was recently saved
l.readHDF()

#evaluation range, defines: blanking time and window length
evran = [35, 55]
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

#time domain x and y data
tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

#correcting a offset in the time domain by subtracting the mean
tdy_mean = tdy-np.mean(tdy)

#fft of the corrected time domain data
fdyl = fftshift(fft(tdy_mean,axis=0),axes=0)

#fft freq and fft shift is here used to scale the x axis (frequency axis)
fdxl = fftfreq(len(fdyl))*l.sra/1e6
fdxl = fftshift(fdxl)

#plot time domain data
plt.figure(1);
plt.plot(tdx,tdy_mean)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

#add LO frequency
lof=l.HDF.attr_by_key('lof')

for i in range(0, len(fdxl)):
    fdxl[i] = fdxl[i]+lof[0]/1e6

#select interesting part of the spectrum
shifter = 12
stopper = 270

#here the right side of the spectrum is selected
y=abs((fdyl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]))
x=fdxl[int(len(fdyl)/2)+shifter:len(fdyl)-1-stopper]


plt.figure(2);
plt.plot(x, y)
plt.xlabel("f in MHz")
plt.ylabel("Amplitude / a.u.")
plt.show()

print("std rms frequency domain next to peak X: " + str(np.std(y)))
print("MAX of Signal: " + str(max(y)))
```

Figure 128: Screenshot of the used Spin-Echo sequence (signal processing part).

A.5.3 Spin-Echo with phase cycling

Figure 129 shows the pulse generating part of the used Spin-Echo with phase cycling sequence.

```
import numpy as np
from numpy.fft import rfft, irfft, rfftfreq
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq, ifft, ifftshift
import limr
from scipy.signal import hilbert

def phase_shift(signal, angle):

    # FFT calculation
    f_spectrum = fft(signal)
    # phase shift in frequency domain (multiply by
    f_spectrum *= np.exp(-1.0j * np.deg2rad(angle))
    # IFFT -> back to time domain
    phaseshifted_signal = ifft(f_spectrum, n=len(signal))
    return phaseshifted_signal

if __name__ == '__main__':

    l = limr.limr('./pulseN_test_USB.cpp');

    l.noi = -1                                          # hardcoded initialization of the lime. needed if parameters
                                                        # (e.g. Gain, tgi, tdi, are changed and need to be set to chip)
    tgtfreq = 83.56e6                                   # target frequency of the experiment
    if_frq = 1.2e6                                      # IF or base band frequency
    l.lof = tgtfreq-if_frq                              # LO frequency (target frequency - base band frequency)
    l.sra = 30.72e6                                     # Sampling Rate
    l.nav = 250                                         # number of averages
    l.nrp = 1                                           # number of repetitions
    l.tdi = -45                                         # TX I DC correction
    l.tdq = 0                                           # TX Q DC correction
    l.tgi = 2047                                        # TX I Gain correction
    l.tgq = 2039                                        # TX Q Gain correction
    l.tpc = 3                                           # TX phase adjustment
    l.rgi = 2047                                        # RX I Gain correction
    l.rgq = 2047                                        # RX Q Gain correction
    l.rdi = 0                                           # RX I DC correction
    l.rdq = 0                                           # RX Q DC correction
    l.rpc = 1                                           # RX phase adjustment
    l.trp = 5e-3                                        # repetition time
    l.tac = 82e-6                                       # acquisition time (gives minimum buffer size)
    l.t3d = [1, 0, 50, 10]                              # GPIO Pin3 is centered around the pulse
    # phase cycling definitions
    l.pcn = [4, 1]                                      # number of phases
    l.pph = [0, np.pi/2]                                # pulse phase (added to phase shift due to pcn)
    l.pfr = [if_frq, if_frq]                            # pulse frequency
    l.pdr = [3e-6, 6e-6]                                # pulse duration
    l.pam = [1, 1]                                      # relative pulse amplitude
    l.pof = [300, np.ceil(9e-6*l.sra)]                  # pulse arrangement in samples [300] -> 12.5 us offset
    l.npu = len(l.pfr)                                  # number of pulses
    l.rgn = 55.0                                        # RX gain
    l.tgn = 40.0                                        # TX gain
    l.rlp = 3.0e6                                       # RX lowpass
    l.tlp = 130.0e6                                     # TX lowpass
    l.spt =  './pulse/SEPC'                             # directory to save to
    l.fpa = 'setup'

    # call to program
    l.run()
```

Figure 129: Screenshot of the used Spin-Echo with phase cycling sequence (pulse generating part).

Figure 130 shows the signal processing part of the used Spin-Echo with phase cycling sequence.

```
l.readHDF()

#select range which should be investigated (blanking and echo time setting)
evran = [35, 55]
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

tdy_mean = tdy-np.mean(tdy)

#take RX time domain data
tdy_mean_0_90 = tdy_mean[:,0]
tdy_mean_90_90 = tdy_mean[:,1]
tdy_mean_180_90 = tdy_mean[:,2]
tdy_mean_270_90 = tdy_mean[:,3]


#use function to shift RX phases accordingly to kazan (therefore k1, k2, k3, k4)
k1 = tdy_mean_0_90
k2 = phase_shift(tdy_mean_90_90, 270)
k3 = phase_shift(tdy_mean_180_90, 180)
k4 = phase_shift(tdy_mean_270_90, 90)

tdy_mean_self = (k1+k2+k3+k4)

#plot time domain data
plt.figure(1);
plt.plot(tdx, tdy_mean)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

fdy1 = fftshift(fft(tdy_mean_self,axis=0),axes=0)
fdx1 = fftfreq(len(fdy1))*1.sra/1e6
fdx1 = fftshift(fdx1)

#add LO frequency
lof=1.HDF.attr_by_key('lof')

for i in range(0, len(fdx1)):
    fdx1[i] = fdx1[i]+lof[0]/1e6

#cut out interessting part of the spectrum
shifter = 12
stopper = 270

#here the right side of the spectrum is selected
y=abs((fdy1[int(len(fdy1)/2)+shifter:len(fdy1)-1-stopper]))
x=fdx1[int(len(fdy1)/2)+shifter:len(fdy1)-1-stopper]

plt.figure(2);
plt.plot(x, y)
plt.xlabel("f in MHz")
plt.ylabel("Amplitude / a.u.")
plt.show()
```

Figure 130: Screenshot of the used Spin-Echo sequence with phase cycling (signal processing part).

A.5.4 Composite Pulse

Figure 131 shows the pulse generating part of the used composite pulse sequence.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq
import limr

l = limr.limr('./pulseN_test_USB.cpp');

l.noi = -1                                  # hardcoded initialization of the lime. needed if parameters
                                            # (e.g. Gain, tgi, tdi, are changed and need to be set to chip)

tgtfreq = 83.56e6                           # target frequency of the experiment
if_frq = 1.2e6                              # IF or base band frequency

l.lof = tgtfreq-if_frq                      # LO frequency (target frequency - base band frequency)
l.sra = 30.72e6                             # Sampling Rate
l.nav = 250                                 # number of averages
l.nrp = 1                                   # number of repetitions

l.tdi = -45                                 # TX I DC correction
l.tdq = 0                                   # TX Q DC correction
l.tgi = 2047                                # TX I Gain correction
l.tgq = 2039                                # TX Q Gain correction
l.tpc = 3                                   # TX phase adjustment

l.rgi = 2047                                # RX I Gain correction
l.rgq = 2047                                # RX Q Gain correction
l.rdi = 0                                   # RX I DC correction
l.rdq = 0                                   # RX Q DC correction
l.rpc = 0                                   # RX phase adjustment

l.trp = 5e-3                                # repetition time
l.tac = 82e-6                               # acquisition time (gives minimum buffer size)
l.t3d = [1, 0, 50, 10]                      # GPIO Pin3 is centered around the pulse

# phase cycling definitions
l.pcn = [1,4]                               # number of phases
l.pph = [0, np.pi/4]                        # pulse phase (added up with phase due to cycling)

l.pfr = [if_frq, if_frq]                    # pulse frequency
l.pdr = [3e-6, 6e-6]                        # pulse  duration

l.pam = [1, 1]                              # relative pulse amplitude
l.pof = [300, np.ceil(3e-6*l.sra)]          # pulse arrangement in samples [300] -> 12.5 us offset


l.npu = len(l.pfr)                          # number of pulses

l.rgn = 55.0                                # RX gain
l.tgn = 40.0                                # TX gain

l.rlp = 3.0e6                               # RX lowpass
l.tlp = 130.0e6                             # TX lowpass

l.spt =  './pulse/COMP'                     # directory to save to
l.fpa = 'setup'

l.run()
```

Figure 131: Screenshot of the used composite pulse sequence (pulse generating part).

Figure 132 shows the signal processing part of the used composite pulse sequence.

```
l.readHDF()

evran = [28, 48]

evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

print(type(evran[0]))

tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]


tdy_mean_self=tdy
#get rid of dc offset
tdy_mean = tdy-np.mean(tdy)

#load time domain data of the different phases into arrays
tdy_mean_0_45 = tdy_mean[:,0]
tdy_mean_0_135 = tdy_mean[:,1]
tdy_mean_0_m135 = tdy_mean[:,2]
tdy_mean_0_m45 = tdy_mean[:,3]

#add them up
tdy_comp = (-tdy_mean_0_45 + tdy_mean_0_135 - tdy_mean_0_m135 + tdy_mean_0_m45)

fdy_comp = fftshift(fft(tdy_comp,axis=0),axes=0)

fdx_comp = fftfreq(len(fdy_comp))*30.72
fdx_comp = fftshift(fdx_comp)

#plot time domain data
plt.figure(1);
plt.plot(tdx,tdy_comp)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

#add LO frequency to IF data
lof=l.HDF.attr_by_key('lof')
for i in range(0, len(fdx1)):
    fdx1[i] = fdx1[i]+lof[0]/1e6

# shift to interesting part of the spectrum
shifter = 12
stopper = 270

y=abs((fdy_comp[int(len(fdy_comp)/2)+shifter:len(fdy_comp)-1-stopper]))
x=fdx1[int(len(fdy_comp)/2)+shifter:len(fdy_comp)-1-stopper]

plt.figure(2);
plt.plot(x, y)
plt.xlabel("f in MHz")
plt.ylabel("Amplitude / a.u.")
plt.show()

print("std rms frequency domain next to peak X: " + str(np.std(y)))
print("MAX of Signal: " + str(max(y)))
```

Figure 132: Screenshot of the used composite pulse sequence (signal processing part).

A.6 Evaluation of the scaling factor between the arbitrary spectrometer unit and volts

Figure 133 shows the signal generating part for evaluation the scaling factor between the arbitrary spectrometer unit and volts.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq
import limr

l = limr.limr('./pulseN_test_USB.cpp');
l.noi = -1                                  #hardcoded initialization of the lime. needed if parameters
                                            #(e.g. Gain, tgi, tdi, are changed and need to be set to chip)

tgtfreq = 83.56e6                           # target frequency of the experiment
if_frq = 1.2e6                              # IF or base band frequency

l.lof = tgtfreq-if_frq                      # LO frequency (target frequency - base band frequency)
l.sra = 30.72e6                             # Sampling Rate
l.nav = 1000                                # number of averages
l.nrp = 1                                   # number of repetitions

l.tdi = -45                                 # TX I DC correction
l.tdq = 0                                   # TX Q DC correction
l.tgi = 2047                                # TX I Gain correction
l.tgq = 2039                                # TX Q Gain correction
l.tpc = 3                                   # TX phase adjustment

l.rgi = 2047                                # RX I Gain correction
l.rgq = 2047                                # RX Q Gain correction
l.rdi = 0                                   # RX I DC correction
l.rdq = 0                                   # RX Q DC correction
l.rpc = 0                                   # RX phase adjustment

l.trp = 5e-3                                # repetition time
l.tac = 82e-6                               # acquisition time (gives minimum buffer size)
l.t3d = [1, 0, 50, 10]                      # GPIO Pin3 is centered around the pulse


l.pfr = [if_frq]                            # pulse frequency
l.pdr = [21e-6]                             # pulse duration; here set to 21 us which is then evaluated
                                            # by setting the evaluation range around the pulse


l.pam = [1]                                 # relative pulse amplitude
l.pof = [300]                               # pulse arrangement
l.npu = len(l.pfr)                          # number of pulses

l.rgn = 55.0                                # RX gain
l.tgn = 40.0                                # TX gain

l.rlp = 3.0e6                               # RX lowpass
l.tlp = 130.0e6                             # TX lowpass

l.spt =  './pulse'                         # directory to save to
l.fpa = 'setup'


l.run()
```

Figure 133: Screenshot of the script for evaluation the scaling factor between the arbitrary spectrometer unit and volts (signal generating part).

Figure 134 shows the signal processing part for evaluation the scaling factor between the arbitrary spectrometer unit and volts.

```
#reads back the file which was recently saved
l.readHDF()

#evaluation range, set that the pulse defined in l.pdr is evaluated
evran = [13, 33]
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]

#time domain x and y data
tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

#correcting a offset in the time domain by subtracting the mean
tdy_mean = tdy-np.mean(tdy)

plt.figure(1);
plt.plot(tdx,tdy_mean/l.nav)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

print("real rms time domain: " + str((np.sqrt(np.mean(np.square((tdy_mean.real/l.nav))))))
```

Figure 134: Screenshot of the script for evaluation the scaling factor between the arbitrary spectrometer unit and volts (signal processing part).

A.7 Noise Analysis

Figure 135 shows the signal generating part for the noise analysis.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.fftpack import fft, fftshift, fftfreq
import limr

l = limr.limr('./pulseN_test_USB.cpp');

l.noi = -1                                          #hardcoded initialization of the lime. needed if parameters
                                                    #(e.g. Gain, tgi, tdi, are changed and need to be set to chip)
#target frequency of the experiment
tgtfreq = 83.6e6

#IF or base band frequency
if_frq = 1.2e6

l.lof = tgtfreq-if_frq                              # LO frequency (target frequency - base band frequency)
l.sra = 30.72e6                                     # Sampling Rate
l.nav = 1000                                        # number of averages
l.nrp = 1                                           # number of repetitions

l.tdi = -45                                         # TX I DC correction
l.tdq = 0                                           # TX Q DC correction
l.tgi = 2047                                        # TX I Gain correction
l.tgq = 2039                                        # TX Q Gain correction
l.tpc = 3                                           # TX phase adjustment

l.rgi = 2047                                        # RX I Gain correction
l.rgq = 2047                                        # RX Q Gain correction
l.rdi = 0                                           # RX I DC correction
l.rdq = 0                                           # RX Q DC correction
l.rpc = 0                                           # RX phase adjustment

l.trp = 5e-3                                        # repetition time
l.tac = 82e-6                                       # acquisition time (gives minimum buffer size)
l.t3d = [1, 0, 50, 10]                              # GPIO Pin3 is centered around the pulse (used as a Gate Signal)

l.pfr = [if_frq]                                    # pulse frequency
l.pdr = [3e-6]                                      # pulse  duration; short pulse used
l.pam = [1]                                         # relative pulse amplitude
l.pof = [300]                                       # pulse arrangement

l.npu = len(l.pfr)                                  # number of pulses

l.rgn = 55.0                                        # RX gain
l.tgn = 40.0                                        # TX gain

l.rlp = 3.0e6                                       # RX lowpass
l.tlp = 130.0e6                                     # TX lowpass

l.spt =  './pulse'                                # directory to save to
l.fpa = 'setup'


l.run()
```

Figure 135: Screenshot of the script for the noise analysis (signal generating part).

Figure 136 shows the signal processing part for the noise analysis.

```
#reads back the file which was recently saved
l.readHDF()

#evaluation range, here around 20 us of blanking was used to be sure that only noise is measured
evran = [36, 56]
evidx = np.where( (l.HDF.tdx > evran[0]) & (l.HDF.tdx < evran[1]) )[0]


#time domain x and y data
tdx = l.HDF.tdx[evidx]
tdy = l.HDF.tdy[evidx]

#correcting a offset in the time domain by subtracting the mean
tdy_mean = tdy-np.mean(tdy)

plt.figure(1);
plt.plot(tdx,tdy_mean/l.nav)
plt.xlabel("t in µs")
plt.ylabel("Amplitude / a.u.")
plt.show()

print("standard deviation time domain: " + str(np.std(tdy_mean.real/l.nav)))
```

Figure 136: Screenshot of the script for the noise analysis (signal processing part).

A.8 Signal to Noise Ratio dependencies

In [24], a calculation for the expected SNR was performed. Here, the SNR of a FID sequence is used as an example to show that this depends only on the window length used and not on the sample rate.

Taking the FIDs magnitude signal formula from [24], S_{FID} denotes as:

$$S_{FID}[0] = (A \cdot e^{\frac{-t_b}{T_c}} T_c \, f_s + C) \cdot N^{-1} \cdot n \cdot scaling\_factor \tag{50}$$

Where A denotes the amplitude, $T_c = \frac{1}{\frac{1}{T_2} + \frac{1}{T_{2^*}}}$ being the combined time constant, $e^{-\frac{t_b}{T_c}}$ is the decay of the magnetization due to the blanking time t_b, f_s denotes the sample rate, n stands for the number of averages, N stands for the signal length which has to be taken into account as scaling factor as the fourier transform sums up the coefficients and C is a constant due to the asymmetry of the one-sided fourier transform for detailed explanations (please also refer to [24] for detailed explanations). The variable $scaling\_factor$ is the calculated conversion factor between the arbitrary spectrometer unit an volts.

The estimated thermal noise (magnitude) can be calculated as:

$$noise = V_{RMS} \cdot \sqrt{0.429} \cdot N^{-\frac{1}{2}} \cdot \sqrt{n} \cdot scaling\_factor \tag{51}$$

and V_{RMS} is calculated as:

$$V_{RMS} = \sqrt{4kTBR} = \sqrt{4kTf_sR} \tag{52}$$

Where the factor $\sqrt{0.429}$ is needed to convert the noise from a real part into the magnitude, $\sqrt{n}$ as the noise will get higher with the square root of the averages and the $\sqrt{N}$ comes from the conversion from time domain to frequency domain (also here, refer to [24] for detailed explanations). The bandwidth of the formula for V_{RMS}, can also be written as the sample rate f_s, as the receiver will "see" this bandwidth at the input and the fourier transform will be calculated from $-\frac{f_s}{2}$ to $\frac{f_s}{2}$ (note: as mentioned above, a lowpass filter can be used here to reduce the RX bandwidth, but the receiver will still receive the noise over $\pm\frac{f_s}{2}$, it will only be "cut off" by the lowpass filter).

Setting $C = 0$, the SNR can then be calculated as:

$$SNR = \frac{S_{FID}}{noise} = \frac{A \cdot e^{\frac{-t_b}{T_c}} T_c f_s \cdot N^{-1} \cdot n \cdot scaling\_factor}{\sqrt{4kTf_sR} \cdot \sqrt{0.429} \cdot N^{-\frac{1}{2}} \cdot \sqrt{n} \cdot scaling\_factor} \tag{53}$$

And is further:

$$SNR = \frac{A \cdot e^{\frac{-t_b}{T_c}} T_c f_s \cdot \sqrt{n}}{\sqrt{4kTf_sR} \cdot \sqrt{0.429} \cdot \sqrt{N}} \tag{54}$$

now, the terms can be rewritten as:

$$SNR = \frac{A \cdot e^{\frac{-t_b}{T_c}} T_c \cdot \sqrt{n} \cdot \sqrt{f_s}}{\sqrt{4kTR} \cdot \sqrt{0.429} \cdot \sqrt{N}} \tag{55}$$

and they can be further written as ($f_s = \frac{1}{t_s}$):

$$SNR = \frac{A \cdot e^{\frac{-t_b}{T_c}} T_c \cdot \sqrt{n}}{\sqrt{4kTR} \cdot \sqrt{0.429} \cdot \sqrt{t_s \cdot N}} \tag{56}$$

knowing that $t_s \cdot N = T_{acquisition-window}$, the final formula is:

$$SNR = \frac{A \cdot e^{\frac{-t_b}{T_c}} T_c \cdot \sqrt{n}}{\sqrt{4kTR} \cdot \sqrt{0.429} \cdot \sqrt{T_{acquisition-window}}} \tag{57}$$

It can be seen, that the derived formula is not dependent on f_s, but the acquisition window ($T_{acquisition-window}$).

A.9 LimeSDR RX Matching Network for $RX1\_L$ port

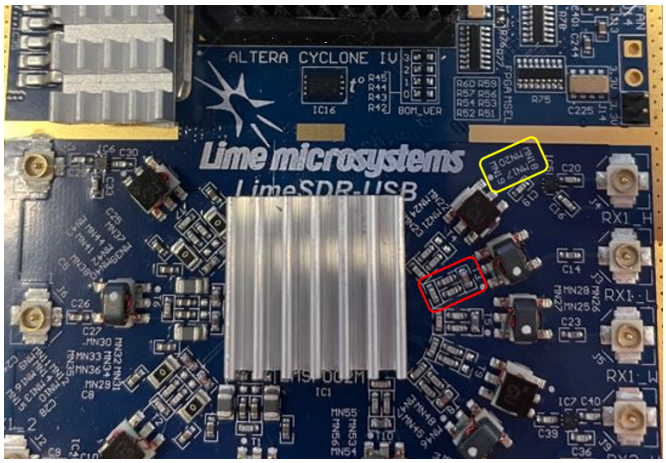

Figure 137: Picture of the RX matching network of the LimeSDR for the $RX1\_L$ port. Red: matching network; yellow: labeling of the components.

References

[1] Carl A. Michal, "A low-cost multi-channel software-defined radio-based NMR spectrometer and ultra-affordable digital pulse programmer". In: Wiley Online Library (2018). url: https://onlinelibrary.wiley.com/doi/epdf/10.1002/cmr.b.21401

[2] Andrin Doll, "Pulsed and continuous-wave magnetic resonance spectroscopy using a low-cost software-defined radio". In: AIP Advances (2019). url: https://aip.scitation.org/doi/10.1063/1.5127746

[3] Andrin Doll, Sourceforge open source project files. url: https://sourceforge.net/projects/limr/

[4] Wikipedia, "Software Defined Radio". url: https://de.wikipedia.org/wiki/Software_Defined_Radio

[5] Siemens AG, "Magnete Spins und Resonanzen - Eine Einführung in die Magnetresonanztomographie". (2003)

[6] Horst Friedbolin, "Basic One- and Two-Dimensional Spectroscopy. 4th Edition". (2005)

[7] Lars G. Hanson, "Is Quantum Mechanics necessary for understanding Magnetic Resonance?" Teaching Resource, (2008). available: https://pfeifer.phas.ubc.ca/refbase/files/LARSG.HANSON-ConceptsInMagnetic-2008-32A-329.pdf

[8] Christoph Aigner, Grundlagen der Biomedizinischen Technik Labor "MR: Simulation, Messung und Auswertung". (2017)

[9] Wikipedia, "Phase Locked Loop". url: https://en.wikipedia.org/wiki/Phase-locked_loop

[10] MyriadRF, LimeSDR hardware description. url: https://wiki.myriadrf.org/LimeSDR-USB_hardware_description

[11] MyriadRF, LimeSDR USB User Guide. url: https://wiki.myriadrf.org/LimeSDR-USB_User_Guide

[12] Lime microsystems, LMS7002M datasheet. url: https://github.com/myriadrf/LMS7002M-docs/blob/master/LMS7002M_Data_Sheet_v3.2r00.pdf

[13] Crowd Supply, LimeSDR. url: https://www.crowdsupply.com/lime-micro/limesdr

[14] RFMD, SPF5189Z Datasheet. available: https://datasheetspdf.com/pdf/761411/RFMD/SPF5189Z/1

[15] NXP Semiconductors, BGY587B Datasheet. available: https://www.mouser.at/datasheet/2/302/BGY587B_3-77227.pdf

[16] Lukas Kaltenleitner, Biomedical Instrumentation Project, "Reflectometer for power monitoring of the transmit channel of a NMR / NQR probe-head". (2021)

[17] H. Scharfetter, M.Bödenler, D. Narnhofer, "A cryostatic, fast scanning, wide-band NQR spectrometer for the VHF range". In: Journal of Magnetic Resonance (2017).

[18] Lorenz Friedrich, Bioimaging and Bioinstrumentation Project, "Dokumentation der Logikschaltung für einen RX/TX-Switch". (2015)

[19] David Astl, Bachelorarbeit, "Integration eines Phase-Cycling-Moduls in ein bestehendes Quadrupol-Resonanz-Spektrometer". (2020)

[20] Karen Sauer et al., "Cancellation of ringing in magnetic resonance utilizing a composite pulse". In: United States Patent Publication (2005). available: https://patentimages.storage.googleapis.com/b6/7d/15/14bc010fdeb068/US20050030029A1.pdf

[21] Bernhard Nißl and Stefan Wunderl, "Analysis of ringing and other artefacts in NQR Spectroscopy". Nuclock Project; Institute of Biomedical Imaging Graz University of Technology. (2021)

[22] Crowd Supply, LimeSDR HF Performance. url: https://wiki.myriadrf.org/LimeSDR_HF_Performance

[23] Lukas Kaltenleitner, Master Seminar, "PCB for extending the output range of 'Arduino Due' on-board DACs". (2021)

[24] Bernhard Nißl, Master Thesis, "SNR optimization for highly sensitive wideband NQR measurements in cryogenic environments". (2022)

[25] Bryan H. Suits, "Nuclear Quadrupole Resonance Spectroscopy" In: Handbook of Applied Solid State Spectroscopy (2006). url: https://link.springer.com/chapter/10.1007/0-387-37590-2_2

[26] Martin Werner, "Nachrichtentechnik, Eine Einführung für alle Studiengänge, 7. Auflage" page 120.

[27] Hermann Scharfetter, Andreas Petrovic, Heidi Eggenhofer, Rudolf Stollberger, "A no-tune no-match wideband probe for nuclear quadrupole resonance spectroscopy in the VHF range". In: IOP Science, Measurement Science and Technology (2014).

[28] Wolfgang Bösch, Michael Gadringer, Lecture Fundamentals of RF and Microwave Engineering Tutorial "T-Anpassungsnetzwerk".