Julia Pfitzer, BSc

NQRduck: Didactic NQR/NMR Experiments through Low-Cost Hardware and Modular Programming

Master's Thesis
to achieve the university degree of
Master of Science
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
Head: Univ.-Prof. Dipl.-Phys. Dr.rer.nat. Martin Uecker

Graz, December 2023

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.

Date	Signature

Abstract

Recent developments in open-source Software Defined Radio (SDR) have enabled the creation of low-cost spectrometers for magnetic resonance applications.

This thesis describes a modular framework for magnetic resonance experiments for didactic purposes using SDR-based spectrometers. It covers the different hardware implementations conducted in the course of this Master's project and gives an overview over their functionality.

Additionally, the thesis introduces NQRduck, a software tool developed during this Master's project. NQRduck is a modular Python framework designed for conducting magnetic resonance experiments. Its architecture allows independent development and installation of modules. The thesis outlines modules used for tuning and matching of probe coils, pulse programming for SDR spectrometers, and executing single frequency and broadband experiments.

The Results section showcases the different functionalities implemented in the course of this project.

Finally, the various limitations and potential future implementations for the NQRduck software and hardware implementations are discussed.

Keywords: Open Source, Magnetic Resonance, Python, Software Defined Radio, Automatic Tuning and Matching

Kurzfassung

Diese Arbeit beschreibt einen modularen Rahmen für Magnetresonanzexperimente zu didaktischen Zwecken unter Verwendung von kostengünstigen SDR-basierten Spektrometern. Sie erläutert die verschiedenen Hardware-Implementierungen, die im Rahmen dieses Projekts durchgeführt wurden, und gibt einen Überblick über deren Funktionalität.

Darüber hinaus wird auf ein im Rahmen des Masterprojekts entwickeltes Softwareprogramm mit dem Namen NQRduck eingegangen. Diese Software ist ein modulares Python-Framework für Magnetresonanzexperimente. Die Funktionalität des Frameworks ist in verschiedenen Modulen gekapselt, die unabhängig voneinander entwickelt und installiert werden können, um die Funktionalität des Programms zu individualisieren. Diese Arbeit gibt einen Überblick über die verschiedenen Module, die für die Impedanzanpassung von mechanischen und elektrischen Probenspulen, die Pulsprogrammierung von SDR-basierten Spektrometern und für Einzelfrequenz- und Breitband-Magnetresonanzexperimente verwendet werden.

Im Ergebnisteil werden die verschiedenen Funktionalitäten vorgestellt, die im Rahmen dieses Projekts implementiert wurden.

Abschließend werden die verschiedenen Limitierungen und potenziellen zukünftige Verbesserungen für die NQRduck-Software und der Hardware-Implementierungen diskutiert.

Schlüsselwörter: Open Source, Magnetresonanz, Python, Software Defined Radio, Automatische Impedanzanpassung

Contents

List of Figures

List of Tables

Acknowledgments

Thank you to the thesis supervisor Hermann Scharfetter for providing me with a great deal of freedom to pursue a wide range of research interests from solid state spectroscopy, cryogenic hardware design to SQUID magnetometry and optics. I enjoyed working with you and always looked forward to fruitful discussions.

Thank you to Lukas and Jan from the Group of Optics of Nano and Quantum Materials at the University of Graz who helped me a lot in a wide range of topics and matters, from refilling helium, to introducing me to various optical spectroscopy methods. I always enjoyed working with you.

Even if they were not directly involved in this thesis, I would like to thank Professor Krenn and Professor Knoll from the University of Graz for fruitful discussions on a wide range of topics. Thank you also for providing the facilities and introducing me to SQUID magnetometry.

Thank you to the Quantum Metrology Group of the Institute of Atomic and Subatomic Physics for trusting me to do research on solid state spectroscopy and magnetometry and providing me with the financial means to do so.

Thank you to the team of the Institute of Biomedical Imaging for the fun time together.

Thank you to my partner for our discussions on software architectures, proofreading the thesis and emotional support during my studies.

Acronyms

ADC Analog-to-digital converter
API Application Programming Interface
ATMV2 Automatic Tuning and Matching Version 2
CAD Computer Aided Design
DAC Digital-to-analog converter
EFG Electric Field Gradient
FID Free Induction Decay
GPIO General-purpose input/output
GUI Graphical User Interface
HV High Voltage
IF Intermediate frequency
I²C Inter-Integrated Circuit
JSON JavaScript Object Notation
LNA Low Noise Amplifier
LO Local Oscillator
LOPRO Logic for Timing Control and Protection
LUT Lookup Table
LV Low Voltage
MRI Magnetic Resonance Imaging
MVC Model View Controller
NMR Nuclear Magnetic Resonance
NQR Nuclear Quadrupole Resonance
OLE Object Linking and Embedding
PA Power Amplifier
RF Radio Frequency
RX Receive
SDR Software Defined Radio
SE Spin Echo
SMA SubMiniature version A
SNR Signal to Noise Ratio
SPI Serial Peripheral Interface
TU Graz Graz University of Technology
TX Transmit
USB Universal Serial Bus
VHF Very High Frequency
VNA Vector Network Analzyer

1. Introduction

Magnetic resonance describes a range of phenomena related to the spin and angular momentum of electrons and nuclei [1]. This field comprises a spectrum of techniques, two of which – Nuclear Magnetic Resonance (NMR) and Nuclear Quadrupole Resonance (NQR) – are the primary focus of this thesis. Despite their similarities in instrumentation, these techniques serve distinct applications and operate based on different principles.

NMR is widely used in diverse fields such as medical imaging and organic chemistry and is based on the splitting of energy levels of ½ spin nuclei in the presence of a static magnetic field.

In contrast, NQR primarily finds application in solid-state spectroscopy. Its functionality is grounded on the ellipticity of the charge distribution within the nucleus, which results in a set of energy levels divided by the interaction between the electrical quadrupole moment and the local electric field.

The key difference in the supporting instrumentation for NMR and NQR lies in the necessity of a strong magnetic field – a requirement for NMR but not for NQR [2].

1.1. Background

Historically, NQR has found numerous applications at the Graz University of Technology (TU Graz), specifically in the research of novel contrast agents for Magnetic Resonance Imaging (MRI) [3]. This has led to the development of a diverse range of spectrometers and hardware by researchers and students [4]. However, these different hardware and software components often demonstrate limited compatibility with each other, complicating the integration of new systems into the existing structure. Functionality, such as broadband measurements, was frequently replicated across separate software programs for each spectrometer, leading to increased maintenance requirements. Improvements implemented in one program necessitated individual updates in each of the others.

In recent years, a new type of accessible hardware for low-cost Software Defined Radio (SDR) based spectrometers has emerged([5] [6] [7] [8]). These spectrometers often cost less than 500€, making them highly accessible for student experiments. One such spectrometer was adapted for NQR experiments from Doll [5] by Kaltenleitner at TU Graz [9]. The spectrometer is based on

the open-source SDR LimeSDR (*Lime Microsystems, Guildford, England* [1]). The LimeSDR-based spectrometer was tested for NQR experiments and relied on programming the pulse sequences by modifying the parameters within Python [2] code.

1.2. Motivation

The motivation for this project was to develop a modular software framework for NMR and NQR experiments that could interface with the existing hardware. The focus was on creating a system that can be used as an educational tool for students. The goal was to make these magnetic resonance techniques accessible to them. Furthermore, it should allow them to easily develop new hard- and software components and integrate them into the existing infrastructure. The previously developed hardware (e.g. probe coils) should be easily usable with the newly developed SDR-based spectrometers.

Additionally, the system should allow the user to perform special experimental setups where the chemical compounds are scanned over a wide range of frequencies. Since the compounds of interest frequently lie within the Very High Frequency (VHF) [10] range of 30-300MHz, the systems should be able to operate in this frequency range.

In distinction to other software for magnetic resonance experiments using SDRs, like the *LimeNMR* program by Franco [6], this project aims to fill a gap in didactic applications for NMR and NQR experiments. It should not rival the likes of large commercial spectrometer manufacturers like Bruker TopSpin [3] or Tecmag, but allow for easy and low-cost magnetic resonance experiments.

The systems should be able to be programmed via a Graphical User Interface (GUI) to allow students to easily perform NMR and NQR experiments. Furthermore, safeguards should be in place in order to prevent students from damaging the system. As an additional point, the software architecture should be modular as to allow an easy integration of additional spectrometers. Finally, the software should be designed in an easy to understand, approachable manner. The software itself is called *NQRduck*.

1.3. Thesis Goals

The project was initiated with primary and secondary objectives set during its planning phase. The primary objectives served as the baseline requirements for the successful completion of the project, which included:

[1] https://limemicro.com/products/boards/limesdr/
[2] https://www.python.org/
[3] https://www.bruker.com/en/products-and-solutions/mr/nmr-software/topspin.html

- Developing a GUI for magnetic resonance experiments using the LimeSDR based spectrometer
- Implementing automatic mechanical Tuning and Matching for broadband spectrum acquisition
- Implementing automatic electrical Tuning and Matching for broadband spectrum acquisition
- Signal-path switching between the system used for Tuning and Matching of the probe coils and the spectrometer

Simultaneously, a series of secondary objectives were established, ranked from highest to lowest priority. These were meant as goals that would not necessarily have to be implemented in the course of this project, but the project architecture should allow for easy integration of those features into the framework.

1. Integration of a previously designed simulator tool for NQR experiments into the new framework.
2. The option to program a variety of different pulse shapes for magnetic resonance experiments.
3. Implementation of signal processing methods within the GUI for analyzing the acquired experimental data.
4. Implementation of calibration methods for the various components of the instrumentation.

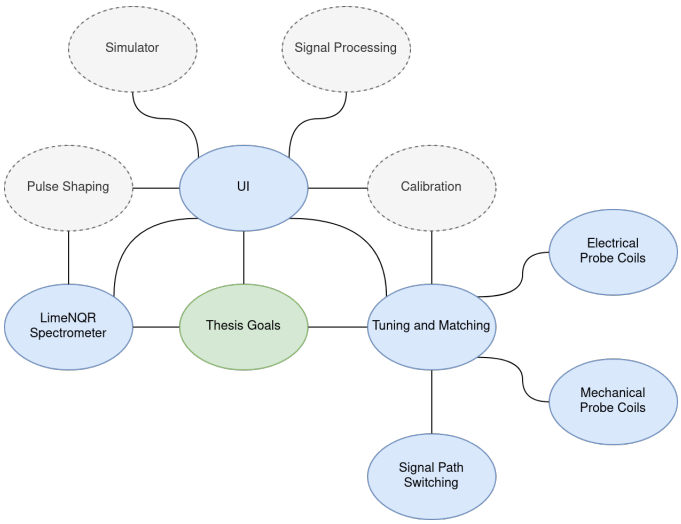


Figure 1.1.: Mind Map of the different project goals depicting the general relationship of the different goals. Optional goals are highlighted in grey while mandatory goals are marked in blue.

1.4. Thesis Overview

The methods section of the thesis covers the physical background behind NMR experiments and highlights the difference to NQR experiments. From the physical background, it establishes what kind of system is necessary for magnetic resonance experiments. The focus is on the different instrumentation for the experiments. The software architecture that integrates all of the different components is then explained. Afterwards, the different modules that make up the whole framework are explained in detail. As a final part of the methods section, the experimental setup for the results section is outlined.

The results section demonstrates the work outlined in the methods section. It focuses on the functionality of the different modules and shows their integration into the overall framework. The discussion section provides an overview about limitations of the different modules and highlights what goals were achieved in the course of this project. It covers a future outlook on what can be improved.

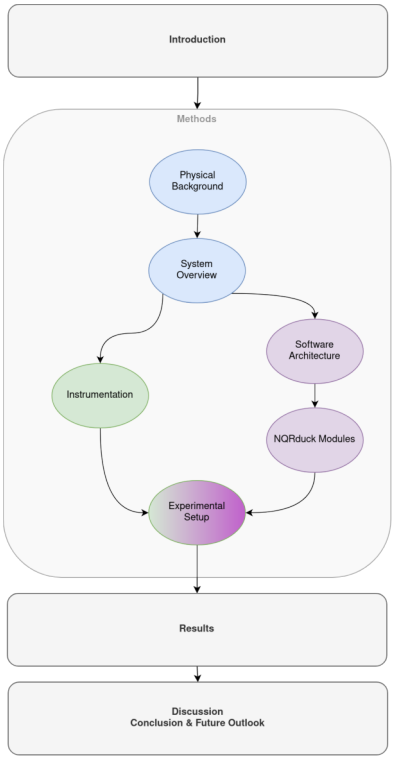


Figure 1.2.: Overview over the topics covered in this thesis.

2. Methods

2.1. Physical Background on Magnetic Resonance

The following explanations on magnetic resonance are based on the book *Introduction to Solid State Physics* by Charles Kittel [1], specifically Chapter 13 on Magnetic Resonance. In addition, the book *Spin Dynamics: Basics of Nuclear Magnetic Resonance* by Malcolm H. Levitt [11] is used as a foundation for the relaxation phenomena and pulse sequences.

2.1.1. Nuclear Magnetic Resonance

NMR is a widely used spectroscopic method in various fields such as organic chemistry and medical imaging, due to its ability to provide detailed information about the molecular structure of analytes [12]. The principles of NMR form the basis for many of the experimental techniques discussed in this thesis.

NMR is performed on isotopes with a non-zero nuclear magnetic moment, which is essential for the operation of NMR. The nucleus' angular momentum $\vec{I}$ is given by $\vec{I}\hbar$ in units of $\hbar$, where $\hbar$ is the reduced Planck's constant. This angular momentum gives rise to a magnetic moment $\vec{\mu}$, associated with the nucleus. This moment can be calculated using

$$\vec{\mu} = \gamma\hbar\vec{I} \tag{2.1}$$

where γ is the gyromagnetic ratio, a nucleus-specific constant that dictates the proportionality between magnetic moment and angular momentum. Physically, it measures how much magnetic moment is generated by a certain amount of nuclear spin [1].

When a static magnetic field $\vec{B} = \vec{z}B_0$ is applied, the energy levels of the nucleus split into $2I + 1$ different states (Figure 2.1). This splitting leads to an energy difference, U, between these states, which is directly proportional to the magnetic field [1]:

$$U = -\mu_z B_0 \tag{2.2}$$

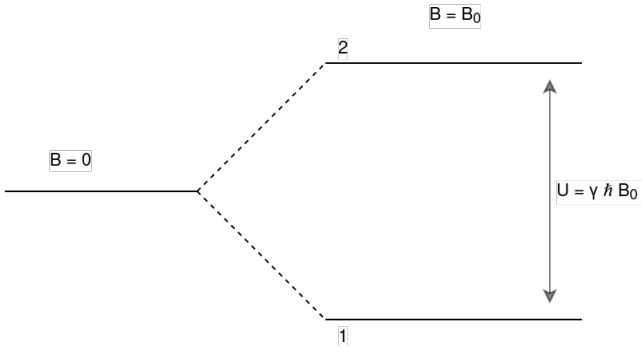


Figure 2.1.: Energy Level Splitting for $I = ½$. The left side of the graphic shows the energy levels when no external magnetic field is applied ($B = 0$). The right side illustrates how the application of a magnetic field ($B = B_0$) leads to a splitting of the energy levels, as described by Equation 2.2. Graphic adapted from [1].

If an Radio Frequency (RF) pulse with energy $E = \hbar\omega_0$ is applied, the criterion for energy absorption in NMR experiments is met. This means that spins absorb energy of an RF pulse and are excited to a higher energy state. The RF pulse is a short burst of electromagnetic radiation at a specific frequency, used to stimulate the transition between nuclear spin states. We can express this frequency as

$$\omega_0 = \gamma B_0 \tag{2.3}$$

This frequency is referred to as the Larmor frequency [11].

Magnetization

In a sample containing a large number of nuclear spins, the individual magnetic moments do not all align perfectly with the applied magnetic field. Instead, due to their thermal motion, they distribute in a manner that their average orientation has a slight alignment with the field. This results in a net magnetization of the sample in the direction of the applied field. This macroscopic magnetization, arising from the average of all individual nuclear magnetic moments per unit volume, is what we measure in NMR experiments.

The nuclear magnetization $\vec{M}$, therefore, can be seen as the sum of all magnetic moments $\vec{\mu}_i$ per unit volume, as

$$\vec{M} = \sum \vec{\mu}_i \tag{2.4}$$

[1].

When no external magnetic field is applied, the individual spins are randomly oriented, resulting in a net magnetization of zero. However, when a static magnetic field B_0 is applied, the spins have a tendency to align themselves along the direction of the magnetic field. This results in a net magnetization in the direction of the magnetic field.

The net magnetization in thermal equilibrium M_0 along $\vec{z}$ can be described as

$$M_0 = N\mu \tanh \frac{\mu B_0}{k_B T} \tag{2.5}$$

, where N is the number of magnetic moments per unit volume, k_B Boltzmann's constant and T the temperature [1].

Relaxation and the NMR experiment

In a typical setup, an RF coil is positioned perpendicular to the static magnetic field B_0. An RF pulse, tuned to the Larmor frequency associated with the energy difference between the nuclear spin states, is then applied. This pulse provides just the right amount of energy to cause transitions between the spin states. As a result, the net magnetization is rotated into the xy-plane. This occurs because the spins enter into superpositions of the base states, which, on average, result in a net transverse magnetization, as depicted in Figure 2.2.

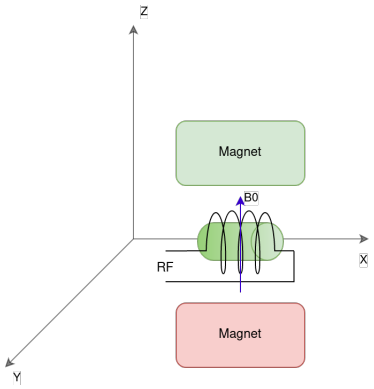


Figure 2.2.: Schematic representation of a typical NMR experiment. The B_0 field is oriented in the z-axis. The sample is indicated in green, and the RF coil, which is placed perpendicular to the static B_0 field, is used to apply an RF pulse. This pulse generates a B_1 field perpendicular to the B_0 field, causing the net magnetization, originally oriented in the B_0 direction, to flip into the xy-plane. Graphic adapted from [1].

After the RF pulse, the system naturally begins to return to thermal equilibrium, a state of lower energy. The precession of the spins now causes a voltage that is re-induced into the coil and can be measured. The signal will of course only be measurable if the spins were excited at the correct frequency. The shape of the re-induced signal will now reveal information about the compound that is being observed [11].

This process is known as relaxation and involves two distinct mechanisms: spin-lattice relaxation and spin-spin relaxation.

Spin-Lattice Relaxation Spin-lattice relaxation describes the process by which the system exchanges energy with its environment, or "lattice". This process causes the net magnetization to return to alignment with the B_0 field, essentially a return to equilibrium following a disturbance, such as an RF pulse. This relaxation process is characterized by the time constant T_1, and is typically seen as an exponential recovery of the longitudinal (z-component) magnetization. It is dependent on various factors and typically occurs in the timescale of microseconds to seconds. This can be described mathematically as

$$M_Z(t) = M_0 \cdot (1 - \exp(-\frac{t}{T_1})) \tag{2.6}$$

where M_0 is the equilibrium magnetization in the z direction [11].

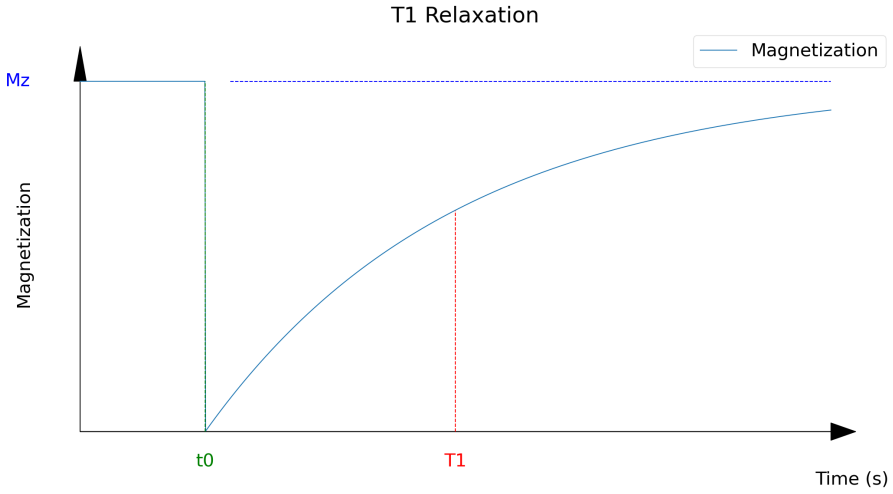


Figure 2.3.: Graphical Representation of T1 Relaxation. The graph illustrates the process of T1 relaxation of a specimen after it is disturbed from thermal equilibrium at time t_0. It is observable from the graph that the magnetization initially drops to zero and then gradually recovers back to M_0 in accordance with equation 2.6. The time 'T1' is specifically marked, which denotes the time at which the magnetization has recovered to 63% of its initial value, M_0.

Spin-Spin Relaxation Spin-spin relaxation refers to the process triggered by fluctuations in the magnetic field. These fluctuations are caused by spin-spin interactions between different atoms, which in simpler terms means that the spins of different nuclei influence each other. These interactions tend to cause dephasing, an irreversible loss of coherence among the precessing spins, leading to relaxation in the xy-plane, often denoted as T2 relaxation [11]. This phenomenon typically occurs in the timescale of microseconds to milliseconds and can be described mathematically as:

$$M_{xy}(t) = M_{xy}(0) \cdot \exp(-\frac{t}{T_2}) \tag{2.7}$$

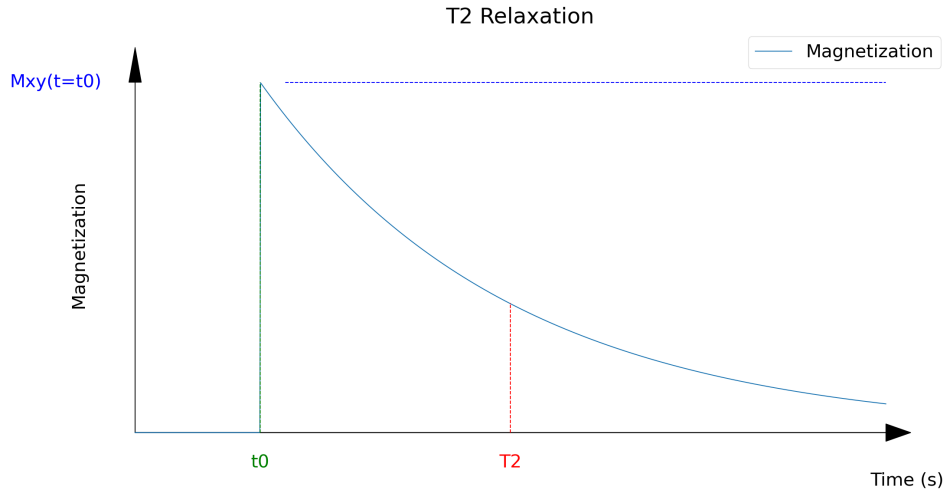


Figure 2.4.: Graphical Representation of T2 Relaxation. The graph illustrates the process of T2 relaxation of a specimen after it is disturbed from thermal equilibrium at time t_0. The graph demonstrates that following the disturbance, the magnetization experiences a decay to zero, rather than a recovery, as per the T2 relaxation phenomenon. This decay is executed according to equation 2.7. The time 'T2' is specifically marked, which denotes the period at which the magnetization has decayed to about 37% of its initial peak value.

Reversible Relaxation In practical NMR experiments, relaxation in the xy-plane can often be observed to occur significantly faster. This is typically due to inhomogeneities, denoted as ΔB_0, in the magnetic field, which can arise from imperfections in the magnet or variations in the sample itself. These inhomogeneities lead to different precession frequencies among different nuclei, causing a type of reversible dephasing often referred to as T2* relaxation.

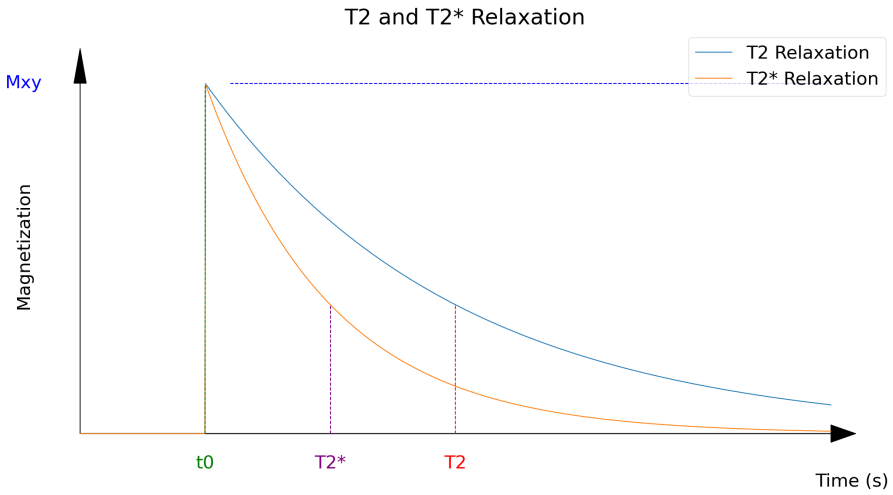


Figure 2.5.: Graphical Comparison of T2 and T2* Relaxation. This graph simultaneously presents the processes of T2 and T2* relaxation of a specimen following a disturbance from thermal equilibrium at time t_0. The T2 relaxation, shown in the blue curve, illustrates the decay of magnetization to zero. On the other hand, the T2* curve shows a more rapid decay due to additional dephasing caused by local field inhomogeneities.

2.1.2. Nuclear Quadrupole Resonance

NQR is a phenomenon observed in nuclei with a spin $I \geq 1$. These nuclei exhibit a non-spherical charge distribution around the nucleus due to their unique spin properties. The ellipticity of this charge distribution is quantified by a value known as the Quadrupole Moment (Q) [2].

The Quadrupole Moment of the nucleus interacts with the electric field of its surrounding environment (Figure: 2.6). This interaction subsequently leads to an energy level splitting into $2I + 1$ distinct states.

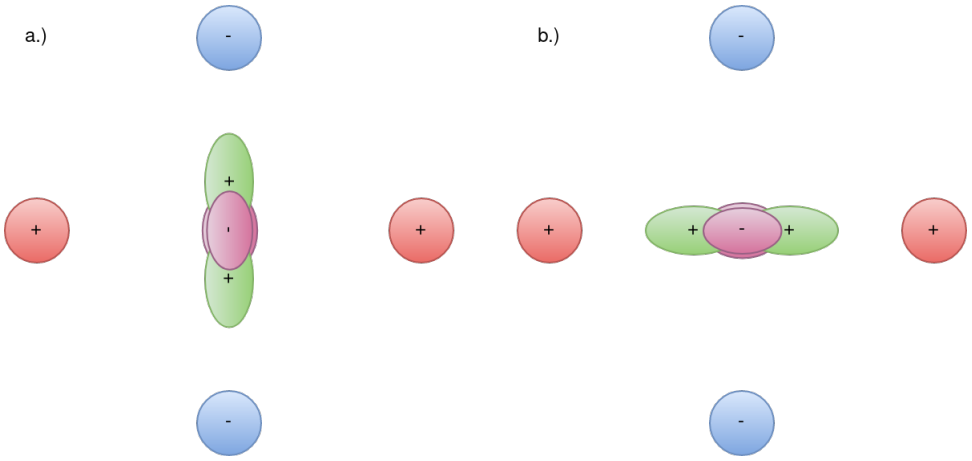


Figure 2.6.: Schematic illustrating energy level splitting in nuclei with non-spherical charge distribution. In panel a.), the nucleus is in the low-energy configuration, while in panel b.), it is in the high-energy configuration. The + and - symbols represent the surrounding charge distribution. Although the overall electric field is zero, the Electric Field Gradient (EFG) is not, leading to configurations with different energy levels for the nucleus with a Quadrupole Moment [2]. Graphic adapted from [1].

When an RF pulse, tuned to the appropriate frequency, is applied, it can lead to interactions among these different energy levels. The resulting re-induced signal can then be observed.

This characteristic of NQR distinguishes it from NMR. Unlike NMR, NQR does not require an external magnetic field to achieve energy level splitting. Instead, the energy level splitting is an intrinsic property of the specimen under examination, depending on the Electric Field Gradient (EFG) produced by the electrons near the nucleus. The splitting is thus modulated by the chemical binding of the quadrupolar atom.

2.1.3. Pulse Sequences

For NQR and NMR, very similar pulse sequences are used that rely on the previously discussed relaxation mechanisms. While there is a wide range of different pulse sequences, only two will be discussed in detail: The Free Induction Decay (FID) and the Spin Echo (SE) sequence.

FID

The simplest form of the NMR experiment is the FID. Here, a single pulse is applied, flipping the magnetization into the xy-plane. Then the exponentially decaying signal is recorded. The pulse is often referred to as $\frac{\pi}{2}$- pulse because it flips the magnetization by 90 °.

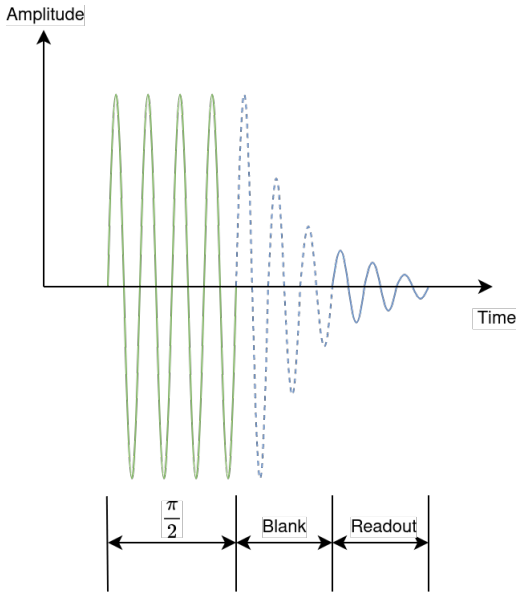


Figure 2.7.: Illustration of a FID sequence. At first, the sample is excited using an RF pulse. The $\frac{\pi}{2}$ denotes that the magnetization flipped by an angle of 90 °into the xz-plane. Afterwards, the signal is a combination of the residual energy of the RF pulse in the resonator (so called 'Coil Ringdown') and the NMR signal. To discriminate between the two signals, a specified time ('Blank') is awaited. The ringdown usually decays faster than the NMR signal, so after waiting, only the NMR signal remains. This signal can then be read by the spectrometer ('Readout') to obtain our NMR signal.

SE

The SE sequence is somewhat more complicated when compared to the FID sequence. The principle is depicted in figure 2.8. This sequence is advantageous if inhomogeneities of the B_0 field cause quick dephasation of the spins because they precess with different Larmor frequencies.

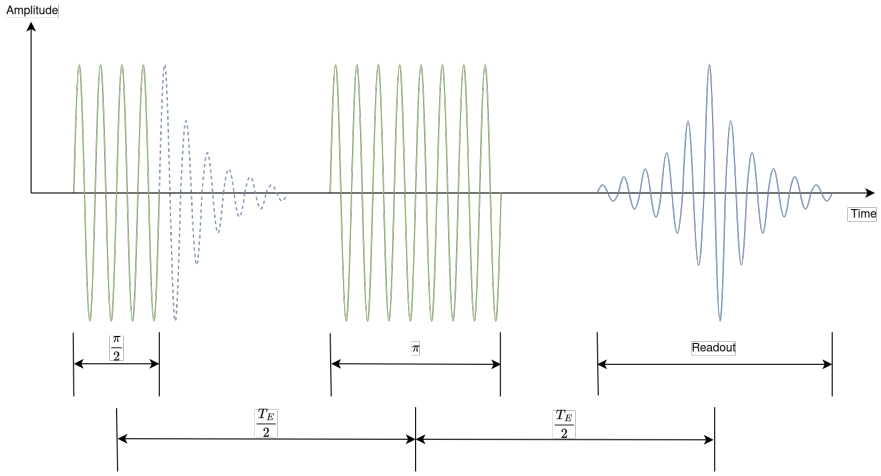


Figure 2.8.: Illustration of a SE Sequence. At first a $\frac{\pi}{2}$ pulse is applied to flip the magnetization in the xy-plane. The signal will decay in the same way as it does for the FID sequence. After a time $\frac{T_E}{2}$ another pulse is applied. This flips the magnetization by 180 °. Rephasing of the different isochromates will now lead to an echo signal after a time T_E. This signal can then be recorded by the spectrometer.

2.1.4. Advanced Techniques

Further pulse sequences can be used to reduce the effect of artifacts or reveal further information about the observed samples. One example for artifact reduction would be Phase Cycling. Here, the phase of different pulse sequences is altered and then sequences are computed in a special way in order to reduce artifacts.

Another pulse sequence that reveals underlying physical principles is the Inversion Recovery Sequence, which helps to measure the $T1$ time of a sample. Here, the magnetization is first flipped by 180°with a π pulse and then a specified time τ is awaited. Then, a $\frac{\pi}{2}$ pulse is applied again, flipping the magnetization back into the xy-plane. This is performed for different values of τ. From this, one can then fit the data for the different pulse sequences to obtain the T_1 relaxation time.

Another example for an advanced pulse sequence would be Composite Pulses, which can be used for artifact reduction [13] [14].

2.2. Physical Background on High Frequency Electronics

2.2.1. Reflection

As we have previously established, the conditions for absorption in magnetic resonance experiments occur at higher frequencies, necessitating a discussion on waveguide effects. The relationship between frequency and wavelength is established as follows:

$$c = \lambda f \tag{2.8}$$

where c is the speed of light in a vacuum, λ is the wavelength, and f is the frequency.

Considering the context of short wavelengths relative to the cable length, i.e. when ($\frac{\lambda}{10} >$ cable length), wave propagation effects must be taken into consideration. One such effect is reflection at locations where there is an impedance mismatch.

The reflection coefficient Γ describes what portion of the forwarded signal is reflected. In the case of magnetic resonance experiments, it can be calculated from the impedance of the probe coil Z_{coil} and the characteristic impedance of the cable Z_0 [15].

$$\Gamma = \frac{Z_{coil} - Z_0}{Z_{coil} + Z_0} \tag{2.9}$$

As can be seen, the reflection of the forward running signal is minimal when Z_{coil} is equal to Z_0.

The reflection coefficient can also be expressed in terms of scattering parameters, specifically the S_{11} parameter [16]. The relationship is given by

$$S_{11} = 20 \log |\Gamma| \tag{2.10}$$

The S_{11} notation will be the primary way of expressing the reflection in this thesis.

2.2.2. One-Port 3-Term Error Model

For the calibration, a simple One-Port 3-Term Error Model is used. For measurements of the reflection coefficient, the measurement has to be calibrated to a calibration or reference plane. Additional cables, different adapters and other changes in system setup will change the results of the reflection coefficient measurements. So by calibrating the system to a reference plane, we can make the measurement more reproducible. This is done by introducing a fictional

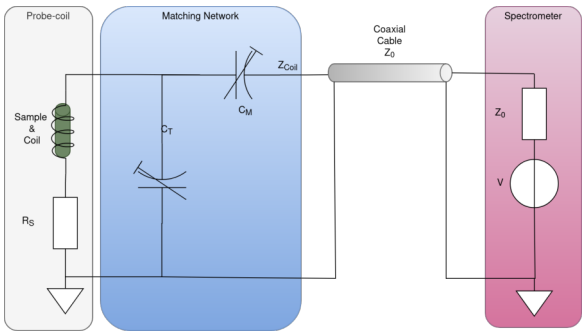


Figure 2.9.: Illustration for Impedance Matching for magnetic resonance experiments. The circuit depicts a parallel resonator topology with capacitive elements for Tuning and Matching. The impedance of the coaxial cable Z_0 is purely resistive (50Ω). The capacitive elements can now be varied in order to achieve a purely resisitive impedance of 50Ω for Z_{coil} which minimizes the reflection coefficient in accordance with 2.13. The figure was adapted from [15].

Error Adapter which adjusts our measured signal according to our reference plane (2.10).

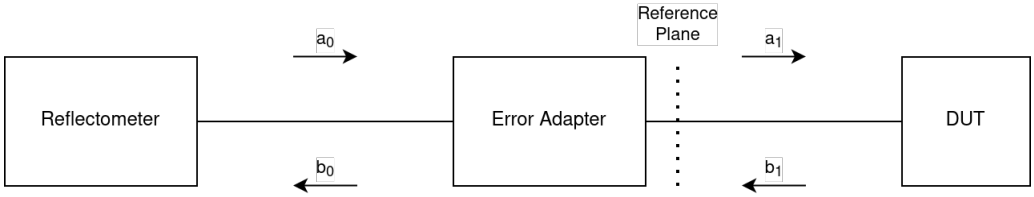


Figure 2.10.: This schematic represents the One-Port 3-Term Error Model. Calibration of the reflectometer is done at the *Reference Plane* with the assistance of the *Error Adapter*. The reflection measurement is taken for a Device Under Test. The term a_0 refers to the incident signal of the reflectometer, while b_0 is the reflected signal measured by the reflectometer. Similarly, a_1 is the incident signal to our one-port Device Under Test, and b_1 corresponds to the reflected signal at our one-port Device Under Test. Figure adapted from [17].

For the One-Port 3-Term Model, three different error terms are assumed to be present in the system according to Figure 2.11.

The term Δ_e can be calculated from the directivity e_{00}, the port-match e_{11} and tracking error $e_{01}e_{10}$ as

$$\Delta_e = e_{00}e_{11} - (e_{01}e_{10}) \tag{2.11}$$

The measured reflection coefficient Γ_M can now be expressed from the different error terms e_{00}, Δ_e and the actual reflection coefficient Γ as following

$$\Gamma_M = \frac{b_0}{a_0} = \frac{e_{00} - \Delta_e\Gamma}{1 - e_{11}\Gamma} \tag{2.12}$$

Since we are interested in the actual reflection coefficient, we can now rearrange equation 2.12 to calculate Γ:

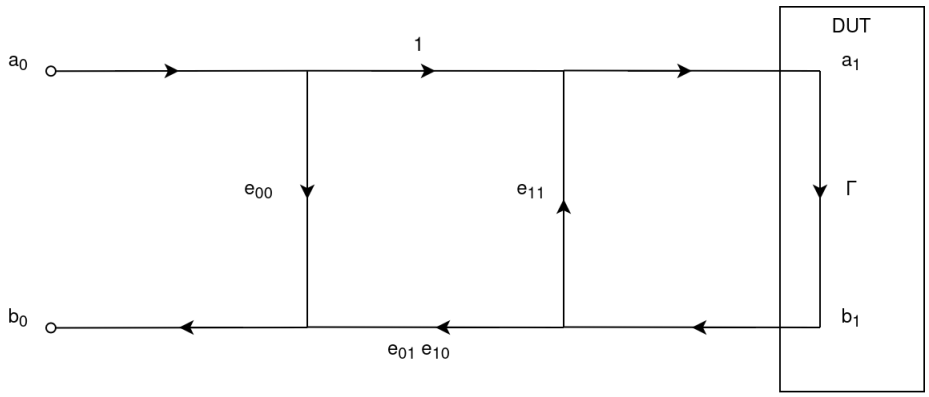


Figure 2.11.: This schematic illustrates the three distinct error terms of the error model. The term e_{00} represents directivity, which describes the degree of isolation between the incident and reflected paths. The term e_{11} corresponds to the port-match, characterizing the degree to which the reflectometer matches the characteristic impedance of the DUT. The term $e_{01}e_{10}$, known as the tracking error, signifies the variations in magnitude and phase shown by the reflectometer across different frequencies. This figure has been adapted from [17].

$$\Gamma = \frac{\Gamma_M - e_{00}}{\Gamma_M e_{11} - \Delta_e} \tag{2.13}$$

To use this equation, we now have to calculate the different error terms for different frequencies. So we bring equation 2.13 into a linear form:

$$e_{00} + \Gamma\Gamma_M e_{11} - \Gamma\Delta_e = \Gamma_M \tag{2.14}$$

We conduct three different measurements to obtain Γ_M with three different DUTs with known reflection coefficients. In practice these measurements are usually chosen as *Short*, *Open* and *Load*.

A *short* measurement refers to a situation where the end of the transmission line is terminated with a short-circuit. Theoretically, in a perfect short-circuit, all of the incident signal power is reflected back toward the source, leading to a reflection coefficient of $\Gamma = -1$.

For an *open* measurement, the end of the transmission line is terminated with an open adapter. Similar to a perfect short-circuit, a perfect open-circuit will also reflect back all the incident signal power, but the phase of the reflected signal is different. So, an ideal open-circuit also has a reflection coefficient of $\Gamma = -1$.

For an ideal *load* of $Z_L = 50\Omega$, the reflection coefficient will be $\Gamma = 0$.

2.3. System Instrumentation

Based on the underlying physical principles, we can establish an experimental flow for NQR and NMR experiments (Figure 2.12).

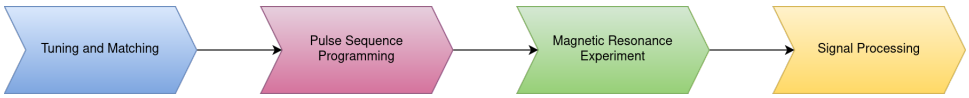


Figure 2.12.: Illustration for the experimental flow of magnetic resonance experiments. After placing the sample in the coil, the probe coil will be tuned and matched to the frequency at which the experiment will be performed ('Tuning and Matching'). Then, a pulse sequence will be selected for the spectrometer, where one can, for example, decide between a FID or SE sequence and set the pulse power ('Pulse Sequence Programming'). Then the experiment will be conducted. Here, the spectrometer will output the programmed pulses and read the magnetic resonance signal ('Magnetic Resonance Experiment'). Finally, the acquired data is further processed using a wide range of signal processing tools ('Signal Processing').

It is evident in figure 2.12 that the magnetic resonance experiment uses both soft- and hardware components.

From 2.3 we know that we have a certain Larmor frequency the experiment will have to be performed at. For our experiments, this frequency is frequently in the range of 35-300MHz. At these frequencies, wave-guide effects such as reflection will take place, necessitating matching of the probe coil impedance to the characteristic impedance of our cables and spectrometer.

The difference in instrumentation between NQR and NMR experiments is depicted in 2.13 and 2.14.

It is noticeable from figures 2.13 and 2.14 that the setups can be used for both types of experiments by simply adding or removing the magnet. Generally, NMR of protons yields higher Signal to Noise Ratio (SNR), has longer relaxation times and needs less power for excitation of the samples [18].

All the different system components will be described in the following sections. It should be noted that the actual hardware implementation for the receive-transmit (RX-TX) switch and impedance matching can be quite different depending on what kind of probe coil is used. The goal of this project is a modular setup where every spectrometer should be able to deal with a wide range of different hardware components.

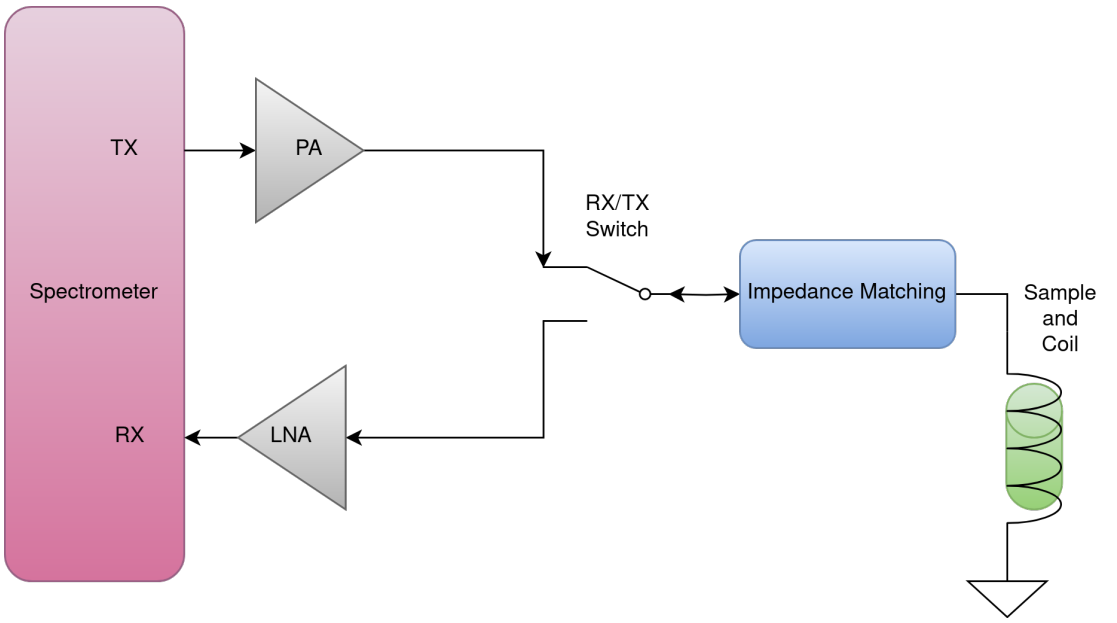


Figure 2.13.: System overview of a typical NQR experiment. The spectrometer outputs a pulse via the Transmit (TX) port, which is amplified by the Power Amplifier (PA). During the transmission, the RX-TX switch connects the PA to the impedance matching circuitry of the probe coil as shown. The Impedance Matching adjusts the impedance of the probe coil and the sample to match the system's characteristic impedance (typically 50Ω). In the RX phase, the RX-TX switch connects the impedance matching component of the probe coil to the Low Noise Amplifier (LNA) to receive the NQR signal.

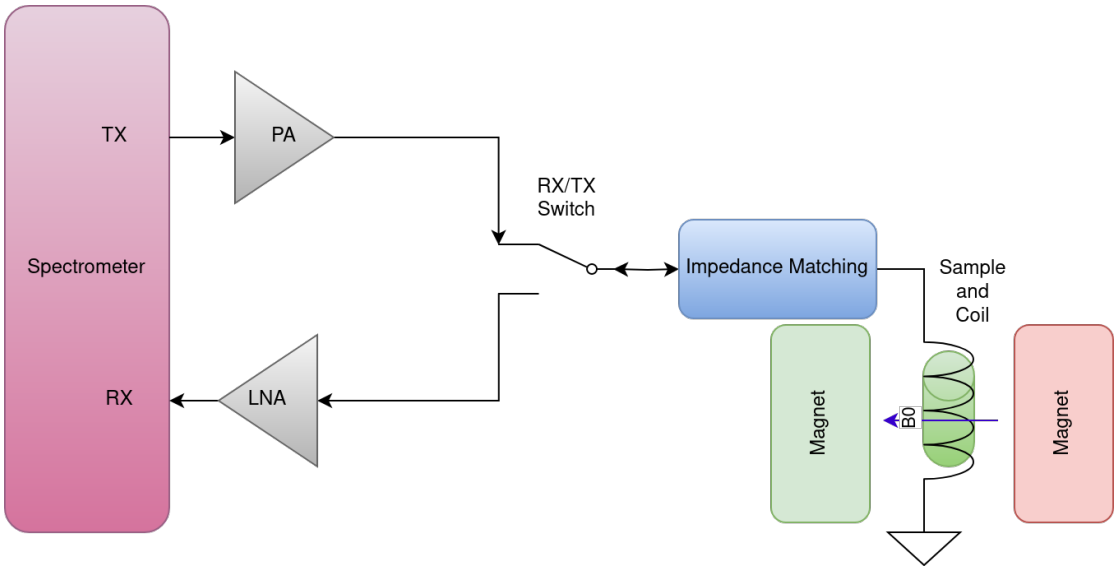


Figure 2.14.: System overview of a typical NMR experiment. The experimental flow is the same as in Figure 2.13. However, in addition the sample is placed in a permanent magnetic field indicated by B_0.

2.3.1. Spectrometer

In a first step, a spectrometer is needed that is able to output the pulse sequences described earlier. The hard- and software background on this has been extensively covered by Doll [5] and Kaltenleitner [9].

In principle, it is based on the LimeSDR (USB Type-A) (*Lime Microsystems, Guildford, England*). It is capable of modulating and demodulating high frequency signals via software [7]. According to the LimeSDR data sheet, it operates in the frequency range from $100kHz - 3.8GHz$ [19]. However, the frequencies available for magnetic resonance experiments are determined by the Local Oscillator (LO) range, because this is the frequency range where RF signals can be generated. For the LimeSDR this is $30MHz - 3.5GHz$ [5].

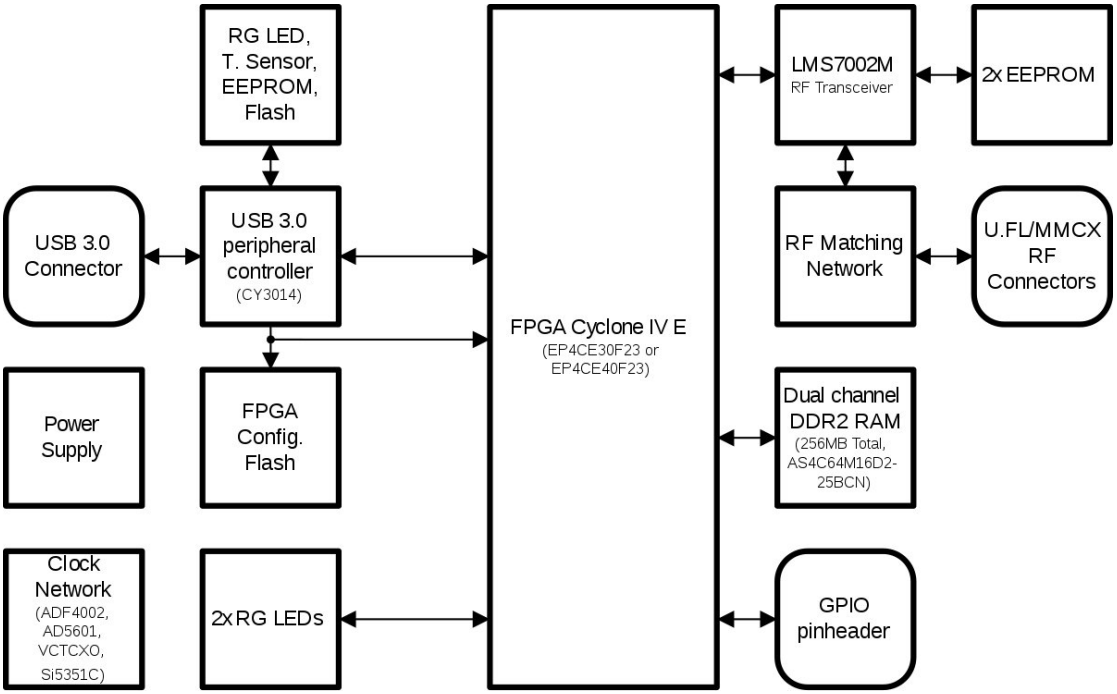


Figure 2.15.: Block diagram of the LimeSDR. The Field Programmable Gate Array in combination with the Field Programmable RF are the heart of the SDR. They can be used to output arbitrary pulse sequences and read the magnetic resonance data. The communication is done via the Universal Serial Bus (USB) 3.0 Connector [5]. Diagram taken from [19].

The software used to control the LimeSDR is provided by Doll [5] [20]. It contains a Python package that can be used for communication with a C++ Application Programming Interface (API). This C++ API handles the direct communication with the LimeSDR board. More details on the software are covered in section 2.5.3.

For this project, the previously developed hardware setup by Kaltenleitner was modified. The previous version included elements needed for Tuning and Matching of electronically tunable probe coils (Figure 2.16).

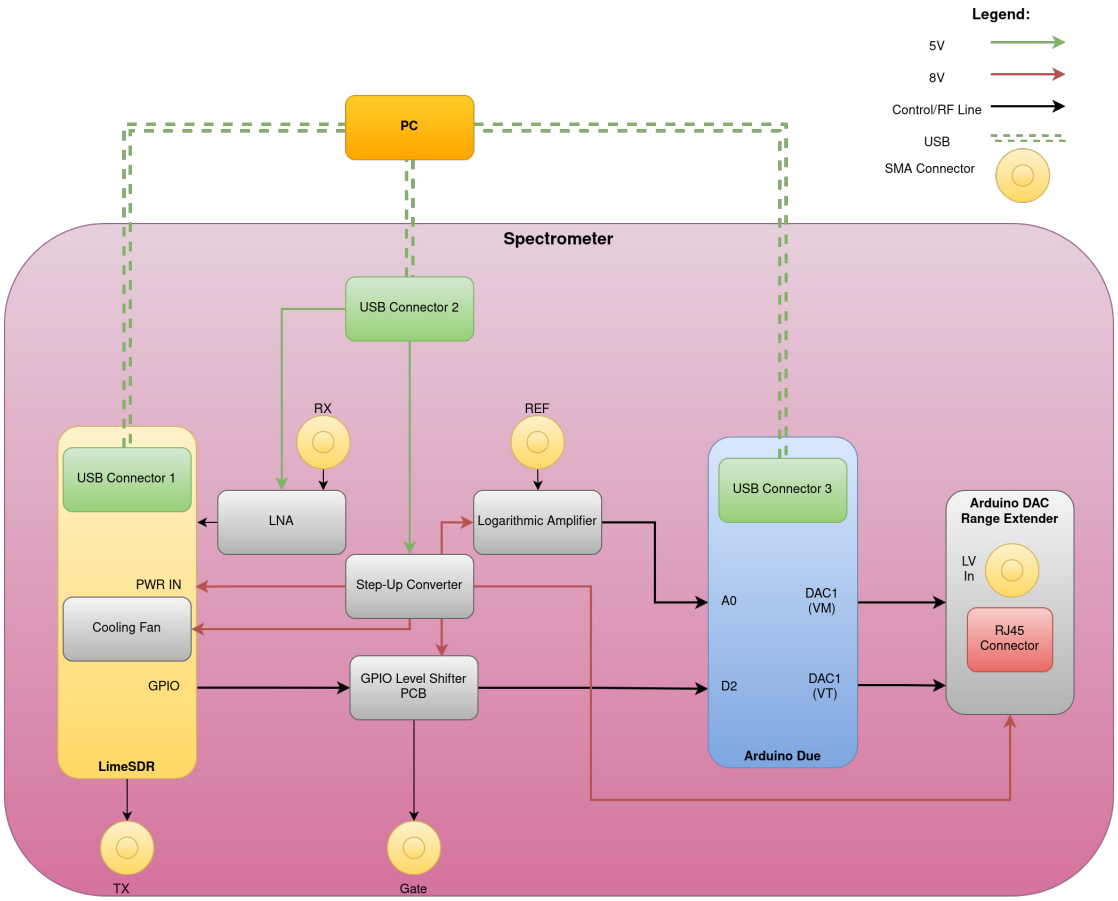
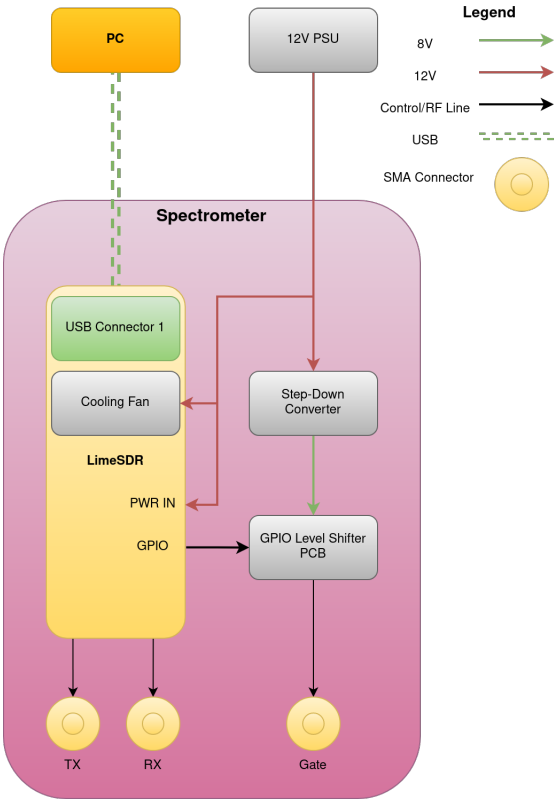
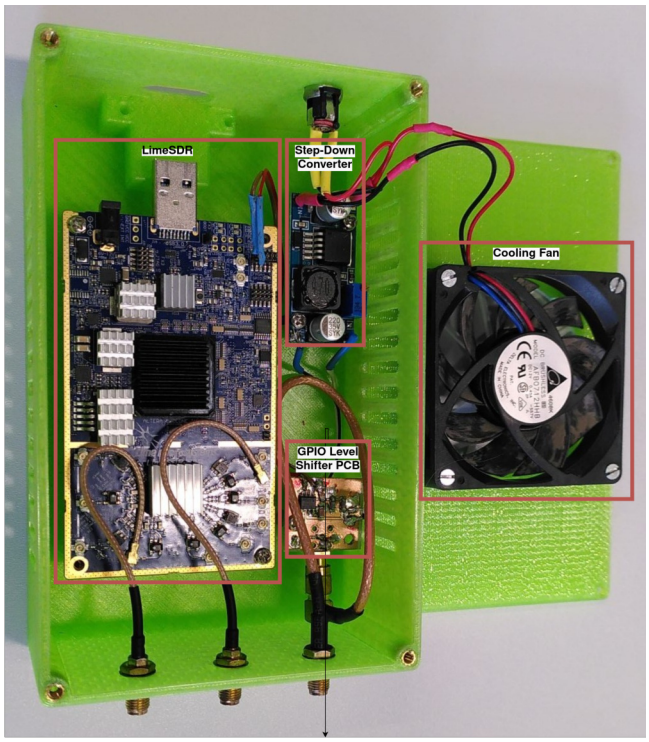


Figure 2.16.: Schematic representation of the previous spectrometer configuration. Components such as the logarithmic amplifier, Arduino Due, and Arduino DAC Range Extender were used for the tuning and impedance matching of the electrical probe coils. Power for various parts of the system was supplied through three separate Universal Serial Bus (USB) connections, as depicted in the schematic.

The setup with 3 separate Universal Serial Bus (USB) connectors confused students who tried to use the spectrometer because they didn't know which cable had to be connected for different kinds of probe coils and experiments. Furthermore, the connectors weren't clearly labeled, e.g. the Arduino Due has two different Micro USB connectors, but only one of them was used. For didactic purposes, the setup therefore had to be adapted to be more user friendly. The new setup is illustrated in figure 2.17a.



a) Schematic illustration of the new spectrometer configuration.



b) Picture of the new spectrometer setup.

Figure 2.17.: Illustration of the new spectrometer configuration. In this new setup, the Arduino Due, Logarithmic Amplifier, and Arduino DAC Range Extender were removed and replaced with specialized Tuning and Matching Hardware described in 2.3.4. This simplifies the system connections to a single 12V power input and one USB connection for PC communication.

In addition to the removal of components used for Tuning and Matching and simplification of the power supply, the components were placed in a 3D printed housing, with labeling of different connections to make usage of the spectrometer more convenient and reduce the mechanical load on RF connectors of the LimeSDR.

Figure 2.18.: Picture of the new spectrometer housing. Different connections are labeled.

A Computer Aided Design (CAD) drawing of the housing can be found in the appendix.

2.3.2. Probe Coils

Since we established that at high frequencies we will observe reflection effects, we will have to minimize said reflection at the Larmor frequency, since this will be the frequency of our pulse sequence.

This can be done with a range of different methods, but here we will focus on three distinct methods.

- Electrically tuned probe coils
- Mechanically tuned probe coils
- Broadband probe coils

Electrically tuned probe coils For electrically tuned probe coils, the Transmit (TX) path of the probe coil is matched over a large frequency range of several MHz via the input matching network in figure 2.19 which represents an approximation of an exponential line. The transmit path is separated from the receive path via PIN diodes which can, as a simplification, be seen as electrically operated switches for high frequencies [21]. The Receive (RX) path is then tuned and matched using varactor diodes which change their capacitance based on an applied DC voltage [22]. One varactor diode is used for tuning and one varactor diode is used for matching of the probe coil, the right combination of the voltages for tuning and matching will lead to a matched impedance of the probe coil at our experiment target frequency (Figure 2.19) [4].

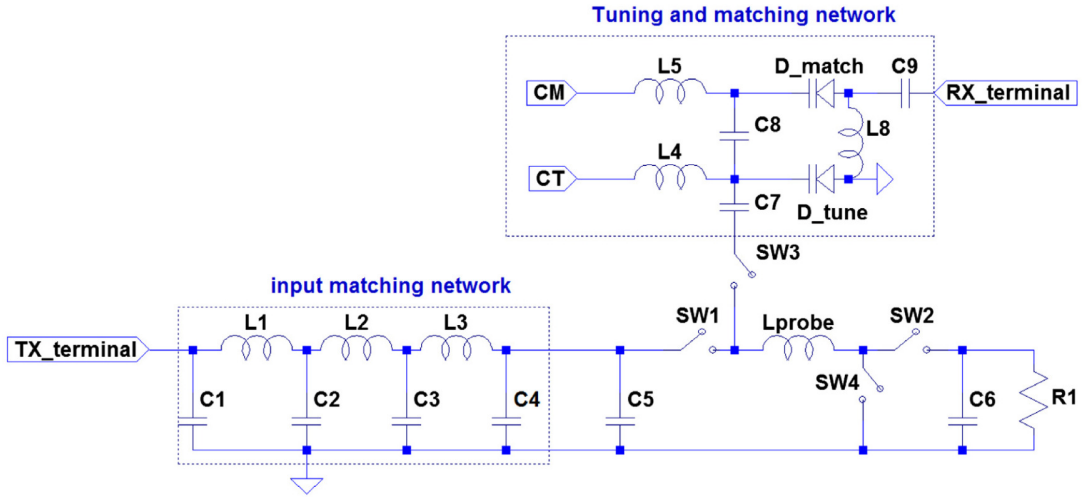


Figure 2.19.: Principle behind the electrically tuned probe coils, taken from [23].

This means that for broadband operation these probe coils need a table where different voltages for tuning and matching corresponding to a certain experiment frequency are saved. Additionally, the operation of these probe coils relies on the usage of the Logic for Timing Control and Protection (LOPRO) described in section 2.3.3 for the RX-TX switch timing and control.

Mechanically tuned probe coils For mechanically tuned probe coils, the capacities used for tuning and matching are changed via mechanical trim capacitors. The capacitors are controlled by two stepper-motors for the tuning and matching capacitors. For operation of the mechanically tuned probe coils, a transcoupler is needed as a passive RX-TX switch as described in section 2.3.3. The transmit and receive path are the same, meaning we operate the coil with a high Q factor for both transmitting the pulses and response signals.

A schematic of the probe coil was presented in section 2.2.1 with figure 2.9

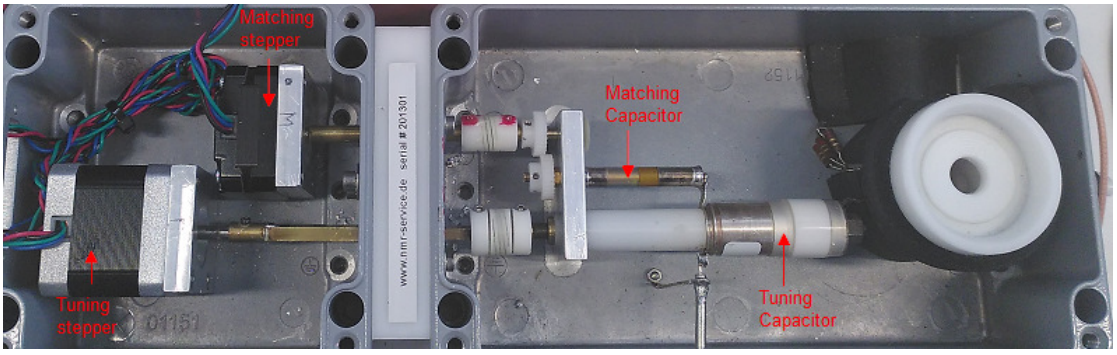


Figure 2.20.: Picture of the mechanically tuned probe coil with labeled components. The picture was taken from [24]

Broadband probe coils Broadband probe coils represent the most straightforward devices for magnetic resonance experiments. Their low Q factor yields a broader bandwidth, enabling coverage of several MHz (typically 5-10MHz) with just one manual Tuning and Matching of the probe coil. However, this approach results in a lower SNR during receive, a notable drawback. These probe coils have been used in broadband measurements of unidentified samples in liquid helium at the TU Graz Institute of Biomedical Imaging. The usage of these devices in liquid helium, with its low temperature of 4K, stems from their simplicity, as they incorporate basic electrical circuits and few mechanical components ([25]). Broadband probe coils rely on the usage of a transcoupler as an RX-TX switch.

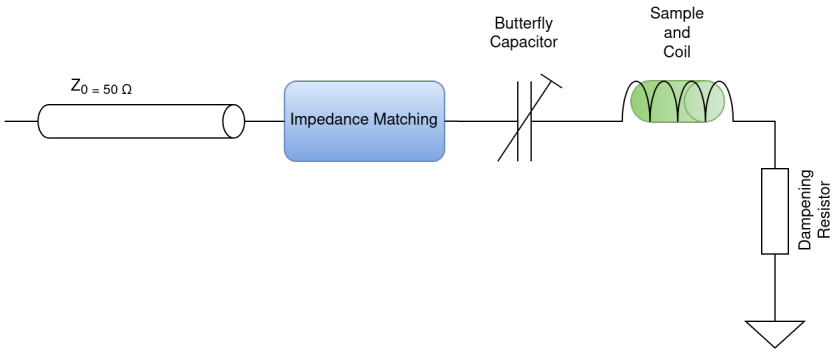


Figure 2.21.: Schematic of the broadband probe coils. The probe coil only has a capacitor for tuning of the probe coil. The dampening resistor in the last implementation had a value of 12Ω. The impedance matching is based on a lumped-exponential line [26].

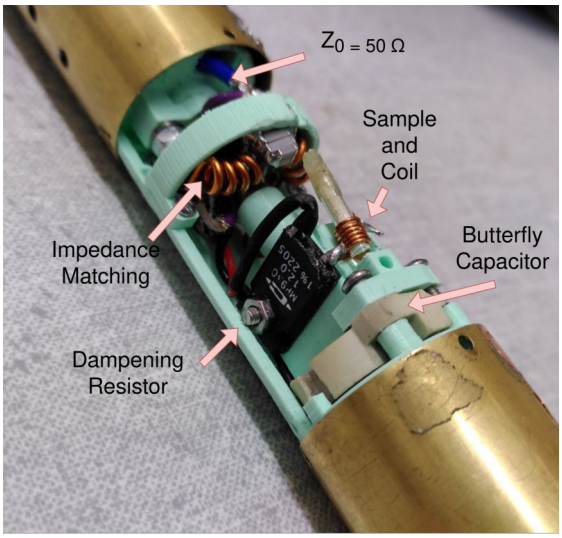


Figure 2.22.: Picture of the assembled probe coil with 3D printed components. A BiPh3Cl2 sample is loaded into the probe coil.

2.3.3. Other Components

Power Amplifier

Since the pulse power output by the spectrometer is not high enough for magnetic resonance experiments, a Power Amplifier (PA) is needed. The PA used for the experiment in this thesis is able to output a power of 100W in the range of 75 to 140 MHz [9]

Low Noise Amplifier

According to Friis' formula, the noise contribution to the measurement chain of the first amplifier in an amplifier chain is the most significant . It is therefore strongly beneficial to use an amplifier with a very low noise figure [15].

For the experiments conducted in this thesis, an off-the-shelf SPF5189Z (*RF Micro Devices, Greensboro, U.S.*) evaluation board was used. A characterization of the Low Noise Amplifier (LNA) was performed by Kaltenleitner [9].

Table 2.1.: Specifications of the SPF5189Z taken from the datasheet and the information printed onto the housing of the LNA.

SPF5189Z Specifications		
Parameter	**Values**	**Unit**
Noise Figure	0.60	dB
Gain	18.7	dB
DC Supply Operation	Single +5V 90mA	
Internal Matching	50 - 4000	MHz

Transcoupler

The transcoupler is a component needed for broadband probe coils and mechanically tuned probe coils. It serves as a passive switch between the TX and RX path.

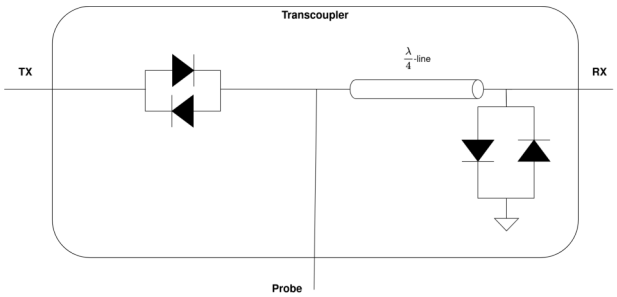


Figure 2.23.: Illustration of the transcoupler's working principle. During the transmit (TX) phase, the diodes conduct, linking the TX and Probe ports, while the receive (RX) port becomes shorted due to the diodes' conductivity. The quarter-wavelength ($\frac{\lambda}{4}$) line transforms this short into an open circuit, causing the RX port to appear open from the TX perspective. In the receive (RX) phase, the non-conducting diodes disconnect the TX port and connect the Probe port to the RX port.

The transcouplers usually have an operating range of several MHz. For this project, the Transcoupler $80MHz - 150MHz$ (*NMR Service GmbH, Erfurt, Germany*) was used. This transcoupler has a more complicated LC-network rather than a $\frac{\lambda}{4}$ line compared to figure 2.23.

LOPRO

The LOPRO is used for the operation of electrically tuned probe coils. It outputs a sequence of pulses that controls the RX-TX switches during magnetic resonance experiments. The LOPRO was thoroughly documented by Friedrich [27]. It also provides voltages for the electrically tuned probe coils where they can be tuned and matched.

When a pulse signal is received at the pulse port, the LOPRO will output a sequence of different signals as can be seen in figure 2.24. The gate signal is used for the PA: Only when the gate is high, the PA will output an RF signal. During transmit, the High Voltage (HV) goes to 350V. This protects the varactor diodes of the electrically tunable probe coil and the RX port of the spectrometer from the high powered pulse because it reverse biases the PIN diodes that connect the RX port and the high powered pulse. In receive mode, the Low Voltage (LV) and HV are both at -5V, connecting the sample coil to the Tuning and Matching network and the RX terminal.

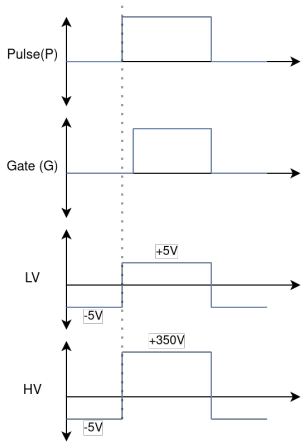


Figure 2.24.: Timing diagram of the LOPRO. Figure adapted from [9][27].

2.3.4. Automatic Tuning and Matching Version 2 (ATMV2)

The ATMV2 system is used for automatic tuning and matching of probe coils. This implementation is an advanced version of a system which was the focus of my bachelor's thesis, designed for the automatic Tuning and Matching of mechanical probe coils using stepper motors [24]. To integrate this system seamlessly within the existing framework and enhance its performance, substantial modifications were made to both the hardware and software.

System overview

The ATMV2 system has two different modes of operation:

- Tuning and Matching of mechanically tuned probe coils (Figure 2.25)
- Tuning and Matching of electrically tuned probe coils (Figure 2.26

The basic principle is the same for both modes.

1. An RF switch connects the ATMV2 system to the probe coil.
2. A reflectometer measures the reflection coefficient of our probe coils at different frequencies.
3. Depending on the probe coil used, different tuning and matching procedures are applied:
    a. For electrically tuned probe coils, voltages will be output to tune and match the probe coil to our experiment frequency.
    b. For mechanically tuned probe coils, the mechanically trimmable capacitors will be adjusted by the ATMV2 system to tune and match the probe coil to our experiment frequency.
4. The RF switch connects the spectrometer to the probe coil.
5. The magnetic resonance experiment will be conducted.

All of the components for both electrically and mechanically tunable probe coils are part of the ATMV2 system. The mode of operation can be specified in the software settings depending on what kind of probe is connected.

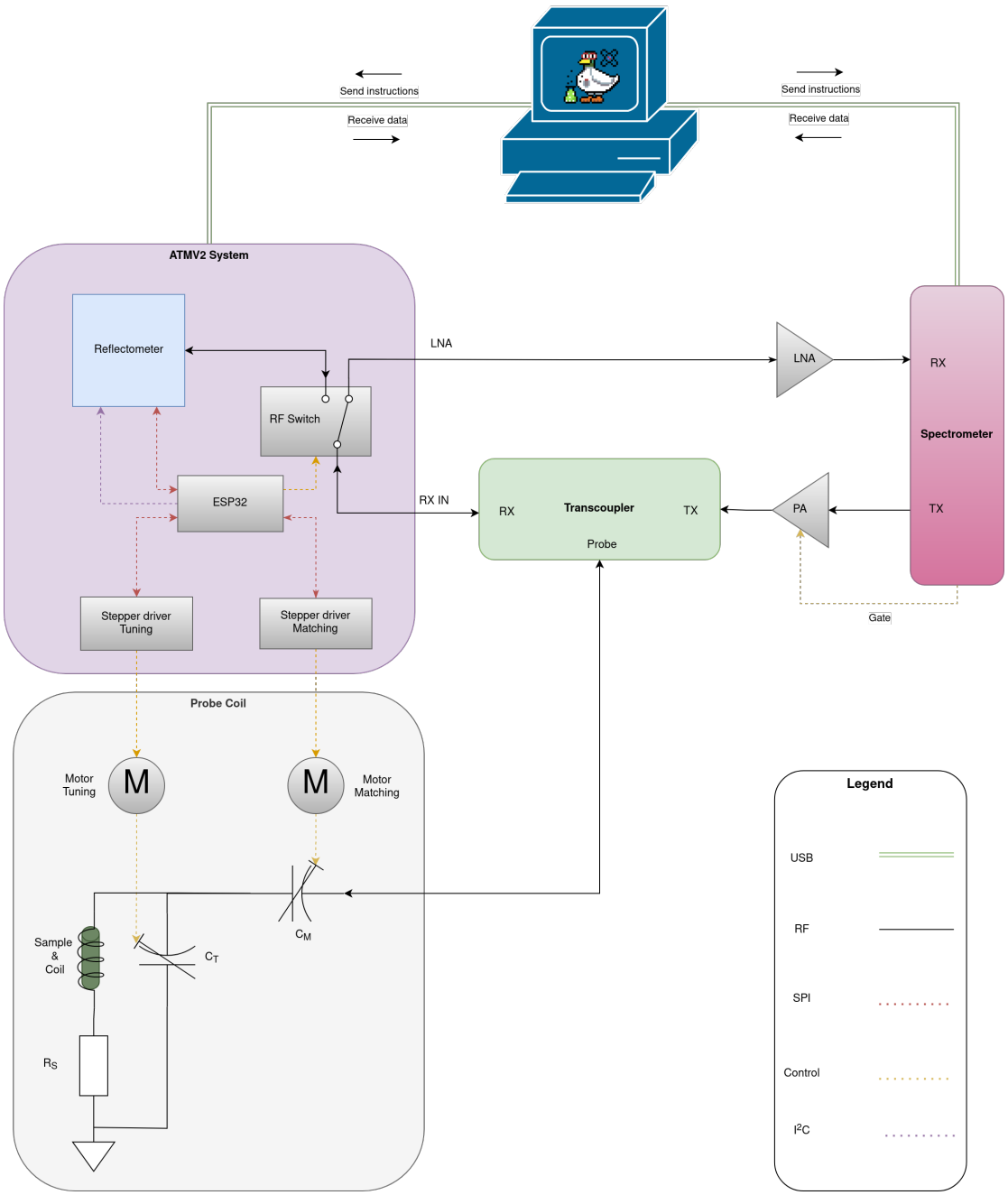


Figure 2.25.: System Overview for the mechanical Tuning and Matching Setup. In addition to a reflectometer, the ESP32 controls two stepper drivers via Serial Peripheral Interface (SPI). These stepper drivers control the steppers that adjust the tuning and matching capacitors. Additionally, the ATMV2 system includes an RF switch that can switch between a mode for tuning and matching of the probe coil and a mode for magnetic resonance experiments. In the depicted image, the RF switch is in the position for magnetic resonance experiments as opposed to the mode used for tuning and matching of the probe coil. The ESP32 communicates with the NQRduck program running on a PC via serial communication. It receives instructions from the NQRduck program and sends data in regards to the reflection measurements and tuning and matching to the NQRduck program.

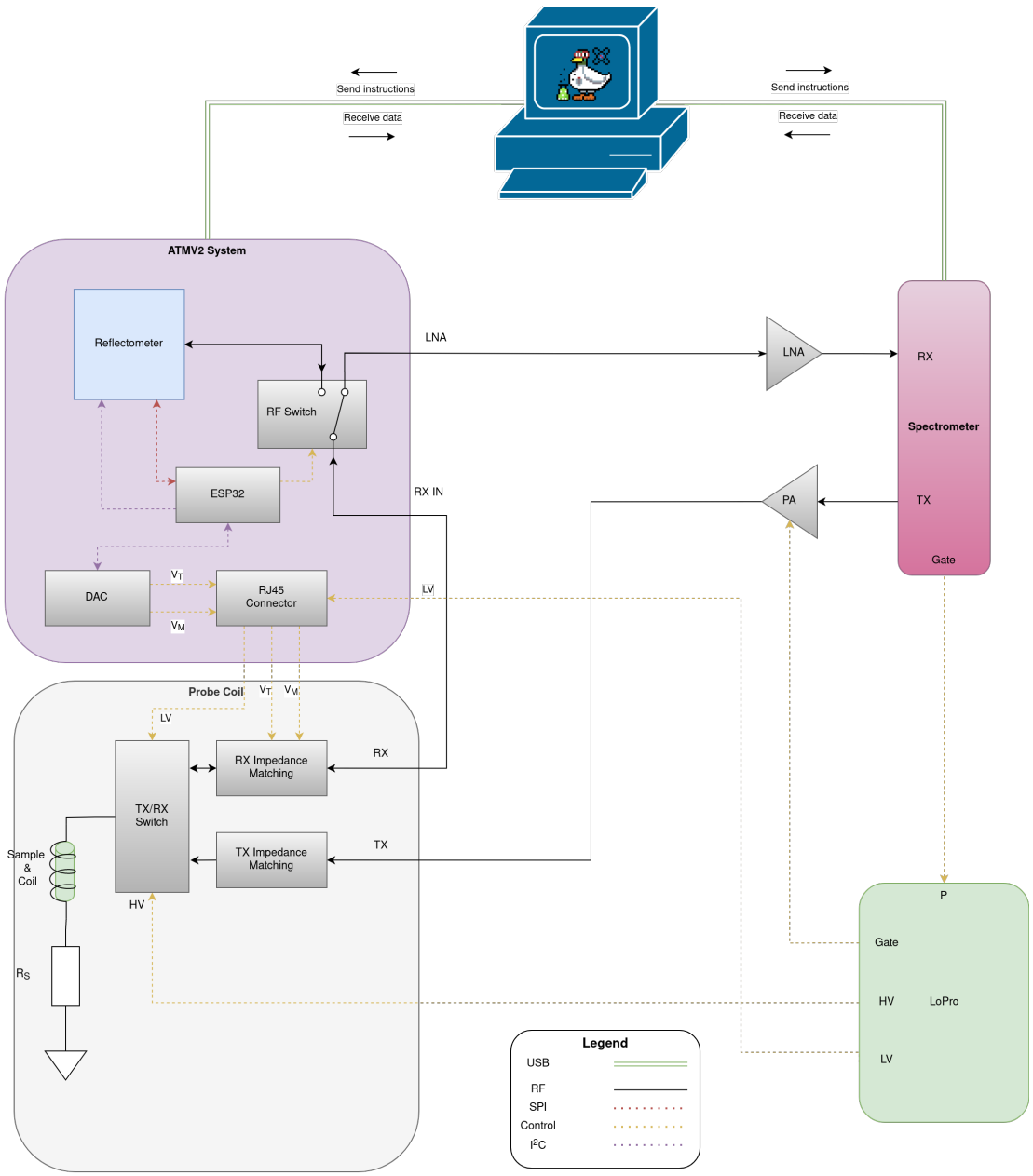


Figure 2.26.: System Overview for the electrical Tuning and Matching setup. The ESP32 uses the ADAC Click module to output voltages to the varactor diodes of the probe coil. By measuring the reflection at different voltage combinations, the probe coil can be tuned and match at the experiment frequency. The LOPRO acts as the control unit for the RX-TX switch of the electrical probe coils. The communication between the ESP32 and the NQRduck is identical to figure 2.25.

Hardware

The hardware is used to measure the reflection of the probe coil at different frequencies. The hardware implementation then allows to control steppers to tune and match mechanically tuned probe coils. Furthermore, the hardware also allows the control of the Digital-to-analog converter (DAC), which outputs a voltage that can be used to tune and match electrically tuned probe-coils.

The hardware was modified in comparison to the bachelor's project [24] because of several reasons.

- The bachelor's project only allowed for the Tuning and Matching of mechanical probe coils. Therefore a DAC had to be added to the setup to allow for Tuning and Matching of electrically tuned probe coils.
- Additionally, an RF switch had to be added, which allowed for switching of the signal pathway between the Tuning and Matching hardware and the spectrometer. It features a simple, ready to use RF switch, which is controlled via a General-purpose input/output (GPIO) Pin of the ESP32.
- Furthermore, the performance of the system was significantly improved by replacing two separate logarithmic amplifiers in combination with an instrumentation amplifier with a single AD8302 off-the-shelve board. In addition to a difference in magnitude between forward and reflected signal, this chip also allows the measurement of the absolute difference in phase. This additionally leads to the possibility to perform a simple one-port calibration of the reflection measurement to a reference plane as described in section 2.2.2.

Reflectometer System Overview

The reflectometer is part of the ATMV2 system as depicted by Figures 2.25 and 2.26 and is used to measure the reflection coefficient of the probe coils at different frequencies. A first iteration of the reflectometer was developed in the course of the Bachelor's project [24]. However, this implementation relied on the usage of two separate logarithmic amplifiers AD8310 (*Analog Devices, Norwood, U.S.*) as described earlier. These logarithmic amplifiers output voltages corresponding to the forward and reflected signal. The difference between the two signals was measured with a instrumentation amplifier.

This implementation had some disadvantages: First, the usage of two separate logarithmic amplifiers meant that the sensitivity might have been different for each amplifier. Additionally, different environmental noise sources might have influenced the logarithmic amplifiers differently, leading to an unwanted differential signal. Furthermore, no information about the phase was obtained with this setup, which made it impossible to apply calibration.

Additionally, the first implementation relied on the ESP32 chip for analog-to-digital conversion, which shows inferior performance to the ADAC Click

board.

In conclusion, the following changes were made to the reflectometer setup:

- Replacement of two logarithmic amplifiers and one instrumentation amplifier with one AD8302 board.
- Additional measurement of absolute phase difference between forward and reflected signal.
- External Analog-to-digital converter (ADC) for improved analog-to-digital conversion.

It should also be noted that the reflectometer is physically integrated into the system for mechanical and electrical Tuning and Matching of probe coils (ATMV2). The ESP32 board also acts as a central controller for all components of the ATMV2 (Figure 2.27).

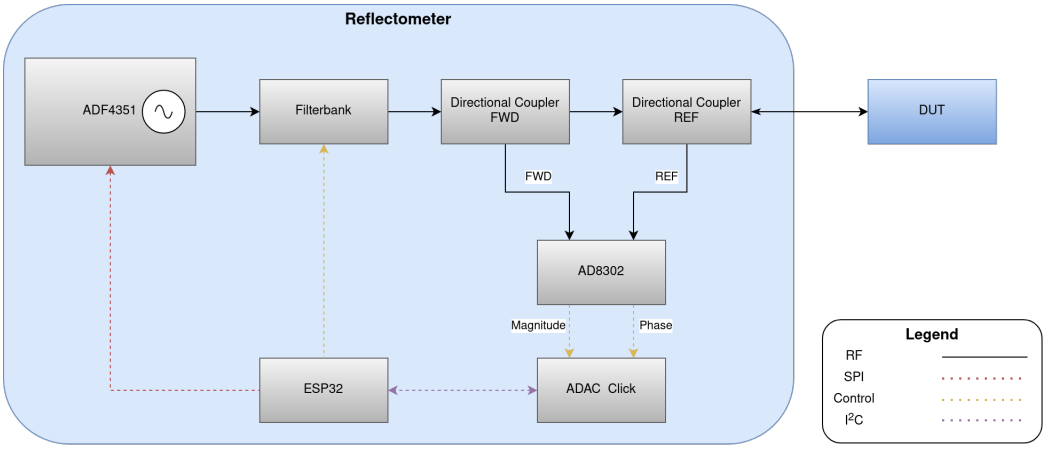


Figure 2.27.: System Overview for the Reflectometer. The ESP32 acts as central control unit for the different components. For a reflection measurement at a single frequency, the ESP32 communicates with the ADF431 frequency synthesizer to output the frequency where the reflection measurement is conducted. Simultaneously, the filterbank is also controlled by the ESP32 to ensure selection of the appropriate filter corresponding to the required frequency range. The first directional coupler now decouples the forward running signal. The second directional coupler decouples the signal reflected by the DUT. The AD8302 now compares the forward and reflected signal and outputs two DC voltages corresponding to the difference in magnitude and in absolute phase between the two signals. The ESP32 then reads these voltages via the ADAC Click. For this, it communicates with the ADAC Click via Inter-Integrated Circuit (I^2C). The read voltages now directly correspond to the reflection coefficient.

Components

An in-depth description of the ESP32 board, Directional Couplers, the ADF4351 and the Filterbank can be found in [24]. The primary focus here will be on the

components that have either been replaced or newly integrated into the setup, such as the ADAC Click, AD8302, and RF switch. Additionally, a short review of the functionality of the other components will also be provided.

ESP32 The ESP32 (*Espressif Systems, Shanghai, China*) DevKitC Board [1] was used as central controller for all components of the ATMV2. Additionally, it handles communication with the NQRduck program running on the computer. A table on the different GPIO connections can be found in the appendix (Table A.1). The ESP32 runs the C++ program used for Tuning and Matching of the different probe coils.

ADF4351 The ADF4351 acts as a frequency synthesizer for reflection coefficient measurements [28]. It is controlled by the ESP32 via Serial Peripheral Interface (SPI). For the control of the ADF4351, a repository by Fanning was modified to be used with the ESP32 [29]. Furthermore, slight modifications were performed to increase the speed while performing frequency sweeps. Different parameters of the ADF4351 are depicted in table 2.2.

Table 2.2.: Selection of parameters relevant for the usage of the ADF4351 taken from the datasheet. The table was taken from [24]

AD4351 Specification		
Parameter	**Values**	Unit
Frequency range:	35 to 4400	MHz
RF Output Power	-4 to 5	dBm
Harmonic Content (Second)	-20	dBc
Harmonic Content (Third)	-10	dBc

.

Filterbank The filterbank is used to filter out harmonics produced by the ADF4351 board. These harmonics limit the accuracy of the measurement of the reflection coefficient. It utilizes two HMC241 switches to switch between different filters [30]. The filters themselves were built in the course of the Bachelor's project and relied on the use of an online tool for calculation of component values [24].

[1] https://www.espressif.com/en/products/devkits/esp32-devkitc

Table 2.3.: Parameters of the filterbank for filtering out harmonic components of the frequency synthesizer's output. Pin A and Pin B are the combination of inputs for the HMC241 switches for each filter. The table was taken from [24]

Filter Name	Cutoff Frequency	Frequency Range	Pin A	Pin B
	MHz	MHz		
FG_71MHz	71	35-70	HIGH	HIGH
FG_120MHz	120	70-120	LOW	HIGH
FG_180MHz	180	120-180	LOW	LOW
FG_280MHz	280	180-280	HIGH	LOW

RF switch The RF switch is used to switch between two different states. Either the RX port of the probe and the ATMV2 system are connected, or the RX port of the probe and the LNA. An off-the-shelf switch was used and is controlled via the ESP32 (Single Positive Control 3.3V). The switch uses the HMC849 *(Analog Devices, Norwood, U.S.)* [31].

Directional Coupler There are two directional couplers used to decouple the forward and reflected signal. Some minor changes to the previous implementation of the ATMV2 were made: The older ZFDC-20-5 (*Mini-Circuits, New York, U.S.*) directional couplers were replaced with ZFDC-20-5-S+ (*Mini-Circuits, New York, U.S.*) directional couplers. The newer directional couplers are smaller and have SubMiniature version A (SMA) connectors.

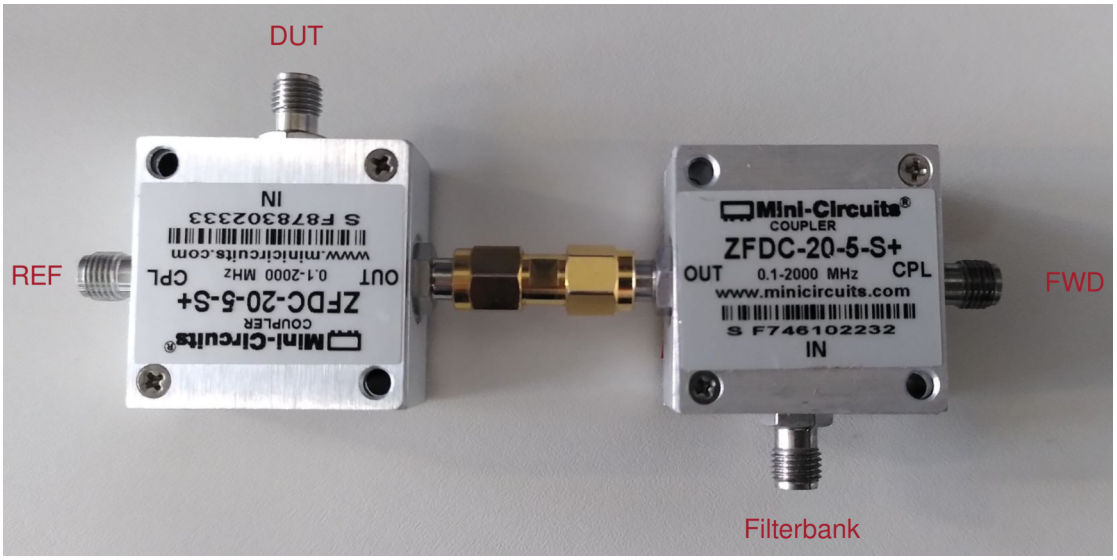


Figure 2.28.: Directional coupler setup with connections labeled. Two seperate directional couplers were used in combination. One of the directional couplers decouples the forward running signal (FWD) and the other coupler decouples the signal reflected by the Device Under Test.

ADAC Click The ADAC Click [2] is used for two different tasks in the context of the ATMV2 [32].

- Analog-to-digital conversion for the reflectometer.
- Digital-to-analog conversion for the Tuning and Matching of electrically tuned probe coils.

This off-the-shelf board is based on the AD5593R [33]. It is controlled by the ESP32 via Inter-Integrated Circuit (I²C). A selection of different parameters can be found in table 2.4.

Table 2.4.: Specifications of the AD5593R taken from the product page [32].

Specification	Description
Type	ADC-DAC
On-board modules	AD5593R 8-Channel, 12-Bit, Configurable ADC/DAC
Key Features	8 12-bit DAC channels, 8 12-bit ADC channels
Interface	GPIO, I²C
Input Voltage	3.3V or 5V

While both the ESP32 ADC and ADAC Click have a resolution of 12 bits, in previous implementations there were significant bugs with the usage of the ESP32 ADC, which limited the speed of frequency sweeps [24]. Furthermore, the ESP32 ADC is highly susceptible to noise [3] which degrades the accuracy of the reflection coefficient measurements.

The ADAC Click was controlled using the AD5593R-Arduino-ESP32-Library by LukasJanavicius via I²C [34]. The library was slightly modified to improve the speed of the analog-to-digital conversion when averaging.

In the context of digital-to-analog conversion, it outputs the appropriate voltages for the tuning and matching varactors. The voltage range for both Tuning and Matching is 0V to 5V.

[2] https://www.mikroe.com/adac-click-click

[3] https:docs.espressif.comprojectsesp-idfenrelease-v4.4esp32api-referenceperipheralsadc.html#minimizing-noise

AD8302 The AD8302 module was off-the-shelf and used for the measurement of the Gain/Loss and absolute phase of the forward and reflected signal [35]. The two AD8310 logarithmic amplifiers in combination with the instrumentation amplifier of the previous system were replaced with the AD8302.

Table 2.5.: Specifications of the AD8302 taken from the datasheet [35].

Specification	Description
Frequency Range	Up to 2.7 GHz
Functionality	Dual Demodulating Log Amps and Phase Detector
Input Range	–60 dBm to 0 dBm in a 50 Ω System
Gain Measurement Scaling	30 mV/dB
Phase Measurement Scaling	10 mV/Degree
Supply Voltages	2.7 V – 5.5 V
Reference Voltage Output	Stable 1.8 V

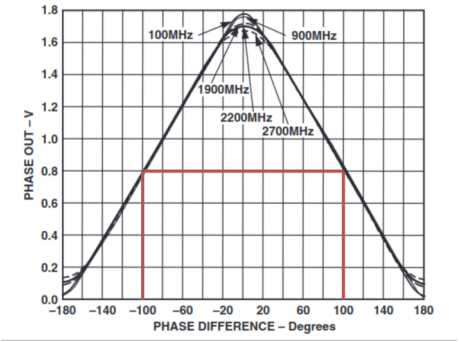


Figure 2.29.: Phase Output (VPHS) vs. Input Phase Difference for different frequencies modified from the data sheet [35]. As can be seen, indicated by the red lines, a phase difference of -100 °leads to the same output voltage as a phase difference of 100 °.

From Figure 2.29 it can be seen that AD8302 can only measure the absolute phase difference. Therefore, a phase correction algorithm had to be implemented. As can be seen, the sign of the slope of the phase measurement is different depending on whether the phase difference is positive or negative. The algorithm was adapted from Johnsen [36]. The pseudocode is given in algorithm 1 and an exemplary phase signal after different steps of the algorithm is visualized in figure 2.30.

Input : Frequency data array, Phase data array
Output: Corrected phase data array

Function `phase_correction`(*frequency_data*, *phase_data*):
  phase_data_filtered ← `MOVING_AVERAGE`(*phase_data*, *WINDOW_SIZE*)
  phase_data_filtered ← `FIX_TRANSIENT_RESPONSE`(*phase_data*, *phase_data_filtered*, *WINDOW_SIZE*)
  peaks ← `FIND_PEAKS`(*phase_data_filtered*)
  valleys ← `FIND_VALLEYS`(*phase_data_filtered*)
  peaks, valleys ← `CHECK_ENDPOINTS`(*phase_data_filtered*, *peaks*, *valleys*)
  frequency_peaks_valleys, peaks_valleys ← `COMBINE_AND_SORT`(*frequency_data*, *peaks*, *valleys*)
  phase_slope ← `CALCULATE_SLOPE`(*phase_data_filtered*, *peaks_valleys*)
  phase_data_corrected ← `APPLY_PHASE_CORRECTION`(*phase_data_filtered*, *phase_slope*)
  return phase_data_corrected

Algorithm 1: Pseudocode for phase correction function.

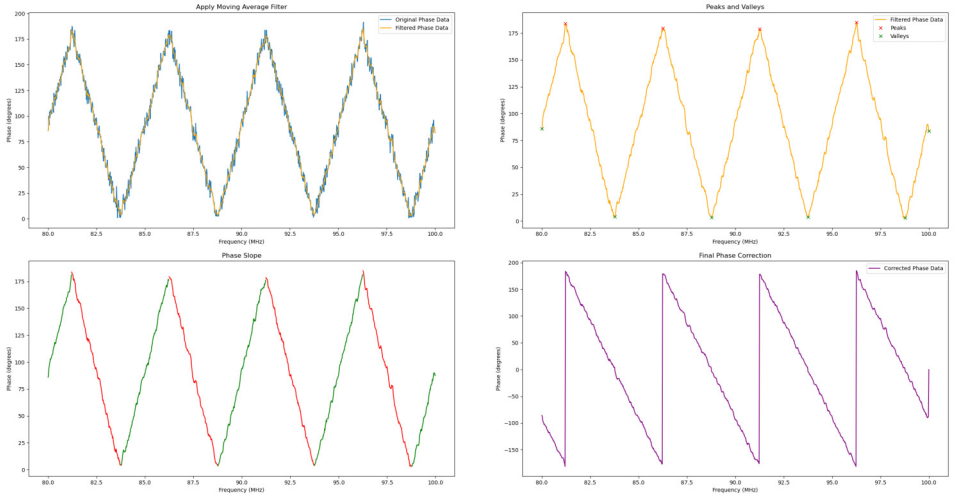


Figure 2.30.: Visualization of the phase correction algorithm with an exemplary signal in the range from 80MHz to 100MHz. At first a moving average filter is applied to the raw data. Then, the peaks and valleys of the signal are identified. From the peaks and valleys, the slope of the signal can be determined. Finally the phase data is corrected using the slope of the signal. A negative slope corresponds to a positive phase, while a positive slope corresponds to a negative phase sign.

Stepper Drivers The TMC2130 (*Trinamic, Hamburg, Germany*) were used as drivers for the steppers [37]. These were chosen because of their sensor-less load detection. This allows for homing of the capacitors used for Tuning and Matching. An in-depth discussion on this can be found in the Bachelor's thesis [24].

Software

The ATMV2 relies on two different programs:

1. A C++ program running on the ESP32 that controls hardware components and implements algorithms for Tuning and Matching.
2. A Python module for the NQRduck framework which handles the GUI and data processing. See section 2.5.1 for more details.

The Python module communicates with the C++ program running on the ESP32 via a USB connection (Figure 2.31). The focus of this section will be on the C++ program since the Python module is explained separately in section 2.5.1.

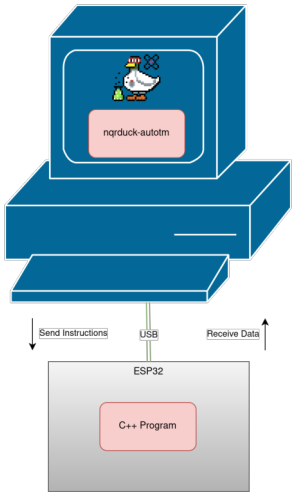


Figure 2.31.: Overview of the different software for the AutoTM Module. The nqrduck-autotm module runs on the PC and is loaded by the NQRduck framework. The C++ program runs on the ESP32. Both programs connect via a serial connection. The nqrduck-autotm sends commands to the C++ program running on the microcontroller.

C++ Program A first iteration of the C++ program was developed in the Bachelor's project [24]. However, because a lot of new functionality was added for the new implementation, the general software architecture was overhauled to allow for a more modular implementation.

The code can be found on GitHub [4].

[4]https://github.com/jupfi/atm

Communication Different commands can be sent to the ATMV2 system using the serial interface of the ESP32. The general structure of such a command is a single character which defines the type of command followed by more arguments defining specific behaviour of the command. The arguments are optional for some commands.

Commands are terminated by a carriage return character (`'\r'`) followed by a newline character (`'\n'`).

Upon receiving a command, the system outputs a confirmation (`'c'`) before executing the instructed tasks.

Software Architecture Different commands are implemented as *Command* classes in the C++ program (Figure 2.32). Depending on the command that is received via serial communication, a *CommandManager* object will call the different commands.

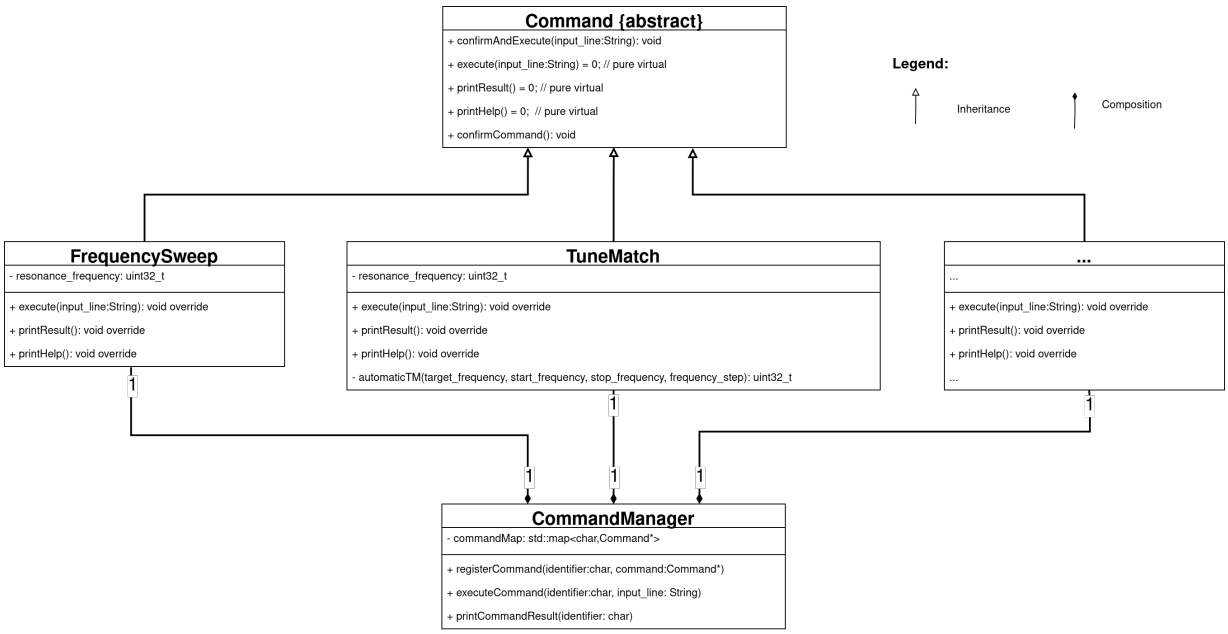


Figure 2.32.: Diagram of the classes used in the C++ program. Depending on what character is sent to the *CommandManager*, a different command will be called. All commands have the same structure as they inherit from the abstract *Command* class.

Summary of Commands A short description of the different commands is given in the following. It specifies the input and output of the different commands and provides and example.

Frequency Sweep (f)

- **Input:** `f⟨start frequency in MHz⟩f⟨stop frequency in MHz⟩f⟨frequency step in MHz⟩`
- **Output During Sweep:** `f⟨frequency in MHz⟩r⟨reflection in mV⟩p⟨phase in mV⟩`
- **Output When Finished**: `r`
- **Input Example:** `f80.0f90.0f0.1`
- **Output During Sweep Example:** `f80.0r1200f678`
- **Description:** Initiates a frequency sweep from the start to the stop frequency in specified increments, returning the reflection and phase in millivolts at each step. The 'r' character signals the completion of the sweep. The input example demonstrates a sweep from 80 to 90 MHz in 0.1 MHz steps. The output example describes a reflection measurement at 80.0MHz whith a reflection of 1200mV and a phase measurement of 678mV.

Tune and Match (d)

- **Input:** `d⟨target frequency in MHz⟩`
- **Output:** `⟨resonant frequency in Hz⟩`
- **Input Example:** `d90.0`
- **Output Example:** `r90000000`
- **Description:** Adjusts the probe coil for optimal resonance at the target frequency and returns the achieved resonant frequency. The given input example sets the target frequency for tuning and matching to 90 MHz. The output example shows that the resonance frequency is at 90MHz after automatic Tuning and Matching of the probe coil.

Homing (h)

- **Input:** `h`
- **Output:** `p⟨position of tuning stepper⟩m⟨position of matching stepper⟩`
- **Input Example:** `h`
- **Output Example:** `p1000m2000`
- **Description:** Resets stepper motors to known reference positions and returns their final positions. The output example reflects the new locations of the tuning and matching steppers after homing (Tuning stepper: '1000', Matching stepper: '2000').

Set Voltages (v)

- **Input:** `v⟨tuning voltage in V⟩v⟨matching voltage in V⟩`
- **Output:** `v⟨tuning voltage in V⟩t⟨matching voltage in V⟩`
- **Input Example:** `v1.5v3.0`
- **Output Example:** `v1.5t3.0`
- **Description:** Sets the specified tuning and matching voltages and confirms the applied settings. In the example, the tuning voltage is set to 1.5V, and the matching voltage to 3.0V. The ATMV2 then sends back the output example as soon as the voltages have been set.

Measure Reflection (r)

- **Input:** `r⟨measurement frequency in MHz⟩`
- **Output:** `m⟨reflection in mV⟩p⟨phase in mV⟩`
- **Input Example:** `r83.5`
- **Output Example:** `m1200p678`
- **Description:** Captures and returns the reflection and phase at the specified frequency. The example command performs the measurement at 83.5 MHz. The output example describes a measured reflection of 1200mV and 678mV for the phase.

Voltage Sweep (s)

- **Input:** `s⟨frequency in MHz⟩o⟨optional tuning voltage in V⟩o⟨optional matching voltage in V⟩`
- **Output:** `v⟨tuning voltage in V⟩t⟨matching voltage in V⟩`
- **Input Example:** `s83.0o2.0o1.5`
- **Output Example:** `v2.1t1.65`
- **Description:** Conducts a voltage sweep at the given frequency, using optional specified voltages. The command selects the optimal voltages for minimal reflection based on either a predefined algorithm or full range sweep, as outlined in Figure 2.33 or Section 2.3.4, respectively. The example initiates the sweep at 83 MHz with pre-defined voltages of 2.0V for tuning and 1.5V for matching. The output example represents the result of the frequency sweep with a tuning voltage of 2.1V and a matching voltage of 1.65V.

Control Switch (c)

- **Input:** `c⟨'p' for pre-amplifier or 'a' for ATMV2⟩`
- **Output:** `c⟨current switch position⟩`
- **Input Example:** `ca`
- **Output Example:** `ca`
- **Description:** Switches the RF switch to connect the probe coil to either the LNA for magnetic resonance experiments or to the ATMV2 for tuning and matching. Outputs the new position of the switch.

The example sets it to ATMV2. The output example describes that the system is now in the 'ca' position.

Move Stepper (m)

- **Input:** `m⟨'t' for tuning or 'm' for matching⟩⟨number of steps⟩,⟨backlash in steps⟩`
- **Output:** `p⟨position of tuning stepper⟩p⟨position of matching stepper⟩`
- **Input Example:** `pt1000,60`
- **Output Example:** `p1000m2000`
- **Description:** Moves the specified stepper by a set number of steps, considering backlash if necessary. The example moves the tuning stepper by 1000 steps and adjusts for 60 backlash steps, with the final position accounting for the backlash correction. The output command describes that the tuning stepper is now at position '1000' and the matching stepper at position '2000'.

Position Sweep (p)

- **Input:** `p⟨sweep frequency in MHz⟩t⟨tuning range in steps⟩,⟨tuning step size⟩,⟨tuning backlash⟩,⟨last tuning direction⟩,m⟨matching range in steps⟩,⟨matching step size⟩,⟨matching backlash⟩,⟨last matching direction⟩`
- **Output:** `z⟨minimum tuning position in steps⟩,⟨tuning last direction⟩ m⟨minimum matching position in steps⟩,⟨matching last direction⟩`
- **Input Example:** `p83.0t40,10,60,-1m500,50,0,1`
- **Output Example:** `z3000,1m1000.-1`
- **Description:** Executes a position sweep to find the minimum reflection positions for tuning and matching steppers. The command accounts for range, step size, backlash, and previous direction of motion, as per the algorithm in Section 2.3.4. The example conducts a sweep at 83.0 MHz with specified parameters for both steppers. The tuning stepper will be moved over a range of ± 40 steps from its current position. It will do this in 10 steps increments while accounting for a backlash of 60 steps. The last direction of the tuning stepper was counterclockwise (-1). The matching stepper will be moved over a range of ± 500 steps from its current position in 50 steps increment. It will not account for any backlash and the last turning direction of the stepper was clockwise ($+1$). The example output describes that the minimum reflection was obtained at a tuning position of 3000 steps and a matching position of 1000 steps. The last turning direction of the tuning stepper was clockwise ($+1$) and the last turning direction of the matching stepper counterclockwise (-1).

Algorithms: Electrically Tuned Probe Coils The focus will first be on the Tuning and Matching of electrically tuned probe coils.

Previous algorithms for Tuning and Matching of the electrical probe coils relied on exhaustive search algorithms [9]. This led to very long search times, because the complexity of the algorithm rises with the square of the number of steps per search.

A reduction of the overall complexity can be achieved by performing a sweep of all different combinations of voltages at a very rough voltage resolution. Around the minimum of the last iteration, the resolution of the scan is increased as described in table 2.6. The algorithm is visualized in figure 2.33.

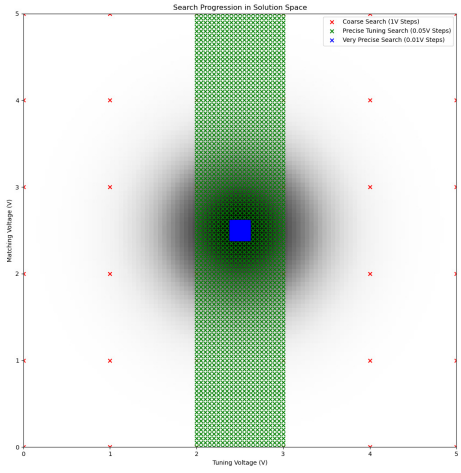


Figure 2.33.: Principle behind the grid search algorithm. The minimum reflection is indicated by the greyscale heatmap with darker values indicating a lower reflection coefficient. The grid gets finer around the mimimum at a voltage combination of 2.5 for tuning and matching voltage respectively. It should be noted that this illustration is a simplification. There is more than one minimum and the Tuning and Matching voltages influence each other significantly.

Table 2.6.: Voltage sweep search parameters. The *tuning_voltage* and *matching_voltage* are the minima for the previous search, respectively.

Search Number	Voltage Step (V)	Tuning Range (V)	Matching Range (V)
1	1.00	0.0 - 5.0	0.0 - 5.0
2	0.05	$tuning\_voltage \pm 0.5$	0.0 - 5.0
3	0.01	$tuning\_voltage \pm 0.1$	$matching\_voltage \pm 0.1$

An additional option for the tuning and matching of electrically tuned probe coils is using pre-defined voltages. This means the probe coil is first manually

tuned and matched to a certain target frequency (e.g. 80MHz). A Lookup Table (LUT) can then be generated using these voltages as a start point for the next exhaustive search. The search is then only performed in a narrow voltage range for the given frequency. The ranges for this are $\pm 0.2V$ for Tuning and Matching voltage in $0.1V$ steps around the pre-defined voltages.

Afterwards, the newly found voltages are used as a voltage starting point for the next frequency. This works because if the frequency only changes a little (approximately 100kHz), as it is the case for the generation of LUTs, the voltages will also only change by a low amount.

The option for using pre-defined voltages is provided in the GUI of the NQRduck AutoTM module.

This option is recommended for high-Q probe coils because the automatic search described in 2.33 might not find a good starting point for the next iteration with coarser grids.

Algorithms: Mechanically Tuned Probe Coils Algorithms for mechanically tuned probe coils are described in [24]. While these algorithms were optimized for speed and robustness, the overall principle stayed the same.

However, for broadband measurements it is also necessary to generate a LUT with different Tuning and Matching positions for different frequencies. An exhaustive search on the full position range, as it is used for the electrically tuned probe coils, would be not possible because of the large number of different stepper positions and high-Q factor of the coil. This means that even for very coarse grids the likelihood of finding a good starting point for the next iteration is minimal.

Thus a two-fold approach is used: First the probe coil is Tuned and Matched to the starting frequency of the LUT. This can either be done manually via the GUI of the NQRduck AutoTM module or using the *Iterative Resonance Tuning* approach described in [24].

From this starting point a, LUT can then be generated using the NQRduck AutoTM module GUI. The algorithm relies on an exhaustive search of different stepper positions around the previous positions similar to the pre-defined voltages for the electrically tuned probe coils.

However, for the LUT to be reliable, a backlash correction had to be implemented for the tuning capacitor. During testing it was noticed that the tuning stepper had an approximate backlash of 60 steps after changing direction. For reference, the tuning steps to change the resonance frequency by 100kHz are about 10-30. This backlash correction is handled by the NQRduck AutoTM module and the C++ program.

2.4. NQRduck Software Architecture

The NQRduck software is programmed in Python. Git was used as a version control system as versioning is essential for larger software projects [38]. Different logos, icons and animations were created using Aseprite[5] (*Igara Studio S.A., Santa Fe, Argentina*).

To allow for easy modification and expansion of the NQRduck software, the architecture was planned accordingly. Functionality is implemented in different modules that can be developed independently of each other. This allows for an independent implementation of the different functionality and makes it less likely for bugs to be introduced into the core application. The modular approach also allows for easy implementation of new modules without in-depth knowledge of the core program or other modules.

Individual features of the software can be installed as separate Python packages, like spectrometer control, pulse sequence programming or simulation of magnetic resonance experiments. The available functionality of the NQRduck program therefore depends on the installed packages.

The Model View Controller (MVC) architecture is a widely established programming pattern for user interfaces. In the MVC architecture the application data is represented in a *Model* object, the GUI in a *View* object and the user input is processed by the *Controller* object [39].

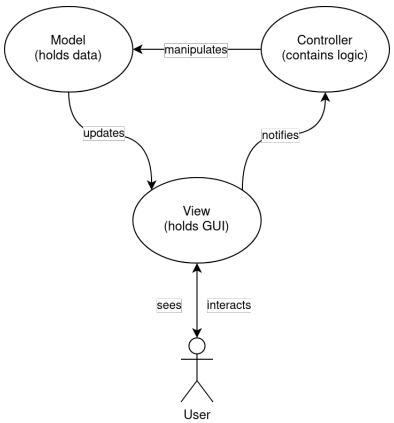


Figure 2.34.: Principle behind the MVC architecture. Figure adapted from [40].

The different components of the thesis and their tasks are shown in figure 2.35.

[5]https://www.aseprite.org/

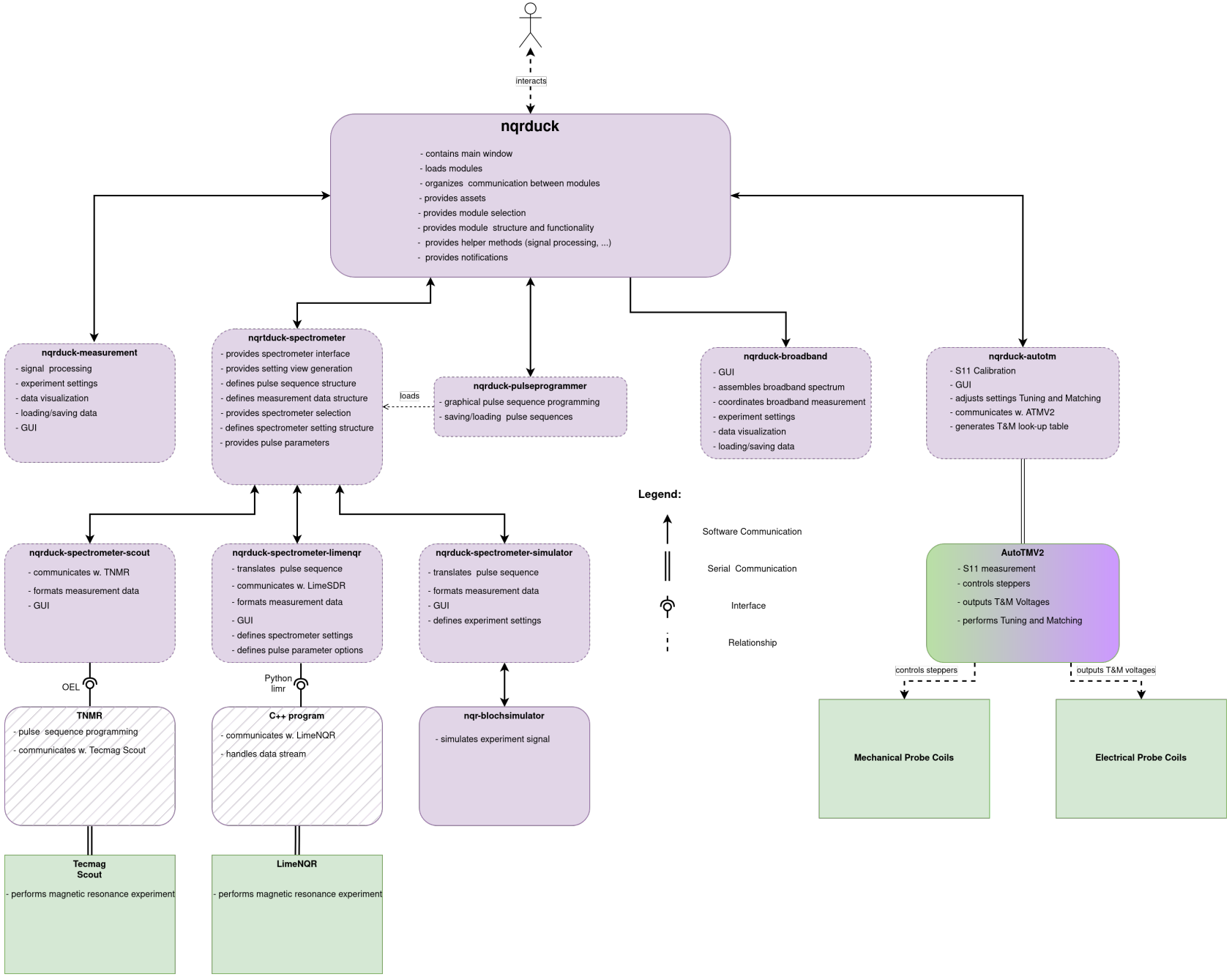


Figure 2.35.: Diagram of the different soft- and hardware components used in this project. Software components are displayed in purple while hardware components are displayed in green. The ATMV2 is composed of hard- as well as software components. The diagram describes the different use cases of the different components and their relationship. The *TNMR* software is third-party software used for the control of the *Scout* spectrometer.

For the NQRduck program, a hierarchical MVC approach was used, where the main application, the so-called *Core*, would have a *MainModel*, *MainView* and *MainController* object. Depending on the user input, the *Core* program then delegates responsibility for user inputs and events to the appropriate modules. It can be seen that both the Core program as well as the different NQRduck modules follow a MVC approach. A class diagram of this structure can be found in figure 2.36.

As can be seen from the class diagram, the *nqrduck* package also provides classes that the *nqrduck-module* will inherit. These classes again follow the MVC structure. Furthermore, the *nqrduck* Core provides classes that provide certain functionality for the different modules. This can range from assets like animations and logos to signal processing methods or unit conversion. Because every *nqrduck-module* can import these functionalities, they only have to be implemented once. If something about this functionality changes, it will only have to be modified in the *nqrduck*.

Within the *nqrduck-modules*, a *Module* class instance is created, encapsulating the corresponding MVC components for that specific module. Here is a Python snippet demonstrating this instantiation process:

```
from nqrduck.module.module import Module
from .model import ModuleAModel
from .view import ModuleAiew
from .controller import ModuleAController

ModuleA = Module(ModuleAModel, ModuleAView, ModuleAController)
```

It should be noted that while the overall class composition in figure 2.36 looks complex, in-depth knowledge of this is not a requirement for developing new *nqrduck-modules*. As long as one adheres to the overall structure of the modules, they will be loaded by the *nqrduck* core. A manual on how new *NQRduck* modules can be developed can be found in the Appendix (B.2).

Furthermore, a template for new NQRduck modules is provided as a git repository [6].

[6]https://github.com/nqrduck/nqrduck-module

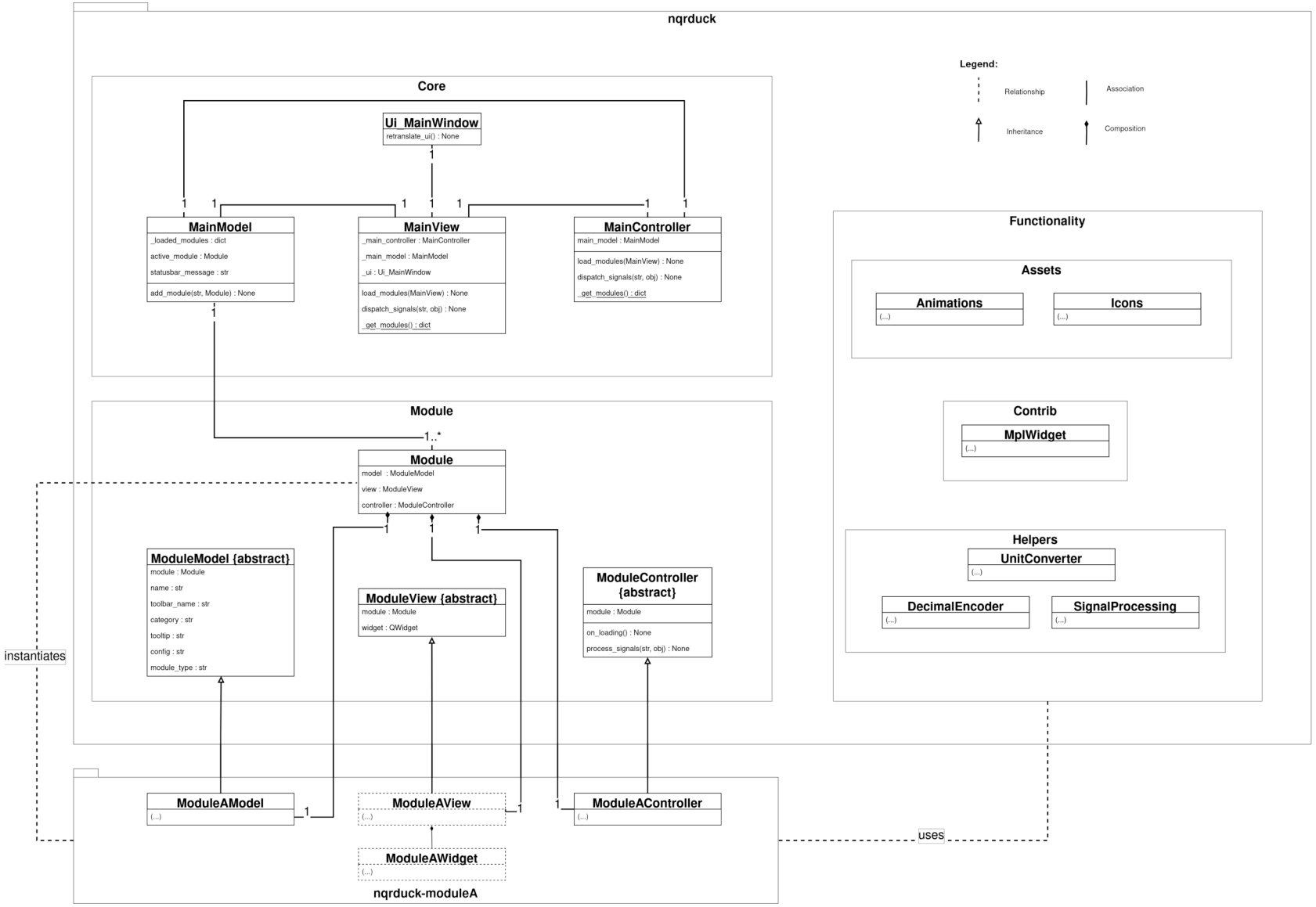


Figure 2.36.: Simplified class diagram of the NQRduck software. The *nqrduck* is an independent Python package. The exemplary NQRduck module *nqrduck-moduleA* is a Python package that implements a *nqrduck-module*. The *nqrduck* package provides the structure for every *nqrduck-module*. The *nqrduck-moduleA* then inherits the *ModuleModel*, *ModuleView* and *ModuleController* classes and implements its functionality there. Then it instantiates a model object with its own module MVC classes. The Core application holds references to all different modules.

2.4.1. UI Structure

The UI is separated into different areas as depicted in figure 2.37. The general structure is that the UI can be developed in separate modules that encapsulate different functionality.

Different Views are generally implemented for each module. The framework for this is PyQt6 [7]. Views can be designed using GUI-based tools like QtDesigner[8] (*The Qt Company, Espoo, Finland*) or can be programmed in Python. These views are then loaded into the main window accordingly.

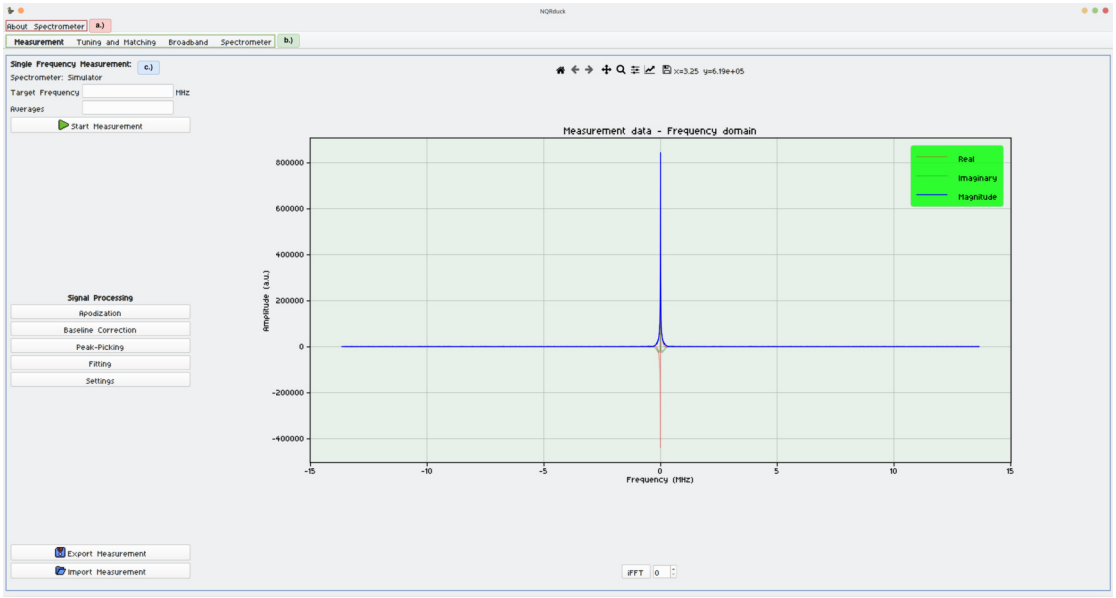


Figure 2.37.: Structure of the UI. The UI is separated into different areas. Section 'a', highlighted in red, represents the menu bar used for general settings of the program and spectrometer selection. Section 'b', outlined in green, allows switching among various modules within the main view of the core, with the active module displayed in bold. Section 'c', depicted in blue, is the active module's view. The currently active module in the figure is the 'nqrduck-measurement' module used for single frequency mangetic resonance experiments. The overall application is part of the NQRduck core and opens when the NQRduck core is started.

As long as the module that is being loaded provides a view, it will be added to the toolbar and it can then be selected from the toolbar as the currently active module.

[7] https://pypi.org/project/PyQt6/
[8] https://doc.qt.io/qt-6/qtdesigner-manual.html

2.4.2. Loading of Modules

On starting of the NQRduck Core, the program utilizes Entry Points, a part of the *setuptools* package [9]. Every *nqrduck-module* therefore needs to specify that it is a module for the NQRduck in its *pyproject.toml* file. An example *pyproject.toml* file can be found in the Appendix B.3. The setuptools package will also resolve dependencies since different *nqrduck-modules* might depend on each other. Additionally, it takes care of external dependencies like *numpy* or *matplotlib*.

Since every module follows the MVC architecture, the core will then attempt to load the MVC objects of the module. Here we strongly adhere to the principle of duck typing. This means that instead of checking if a loaded module is a *nqrduck-module* we check if certain interfaces that every *nqrduck-module* should have can be called [41].

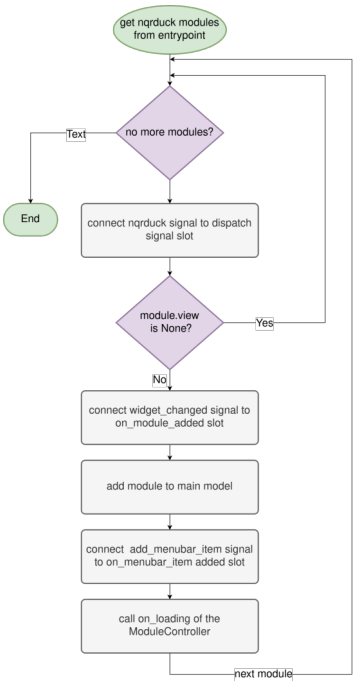


Figure 2.38.: Flowchart of the module loading process

After a module has been loaded, the NQRduck core will attempt to add the module to the UI as described in 2.4.1.

[9] https://pypi.org/project/setuptools/

2.4.3. Communication between modules

Different modules in the NQRduck software architecture communicate via "signals". Signals are specific instructions or data passed between modules, allowing for loose coupling and efficient data flow [42]. Because the communication is handled via signals and slots, different modules don't need references to each other, meaning that they can be developed independently.

1. The communication process begins when a module emits a signal to the core.
2. Upon receiving a signal, the core dispatches it to the different modules.
3. Each receiving module then processes the signal based on its own internal state.
4. (Additionally, objects can be coupled with signals to transmit specific data between modules.)

An example of this communication process is the *'start_measurement*, 100' signal emitted by the 'nqrduck-measurement' module. This signal is triggered when a user interacts with the corresponding interface element, indicating that a measurement should be started at $100 MHz$.

1. The signal is first sent to the NQRduck core, which dispatches it to the different modules.
2. In this case, the 'nqrduck-spectrometer' module is the recipient. This module is designed to handle this type of signal; its controller holds the logic to process it.
3. The 'nqrduck-spectrometer' module contains the data of the currently active spectrometer, which is also implemented as a module and serves as a submodule. An example for this submodule would be the 'nqrduck-spectrometer-limenqr' which is an implementation of the LimeSDR based spectrometer.
4. Upon receiving the signal, the 'nqrduck-spectrometer' module delegates the signal to the corresponding active spectrometer submodule. This submodule is responsible for actually initiating the measurement, as it holds the necessary logic for operating the specific spectrometer.

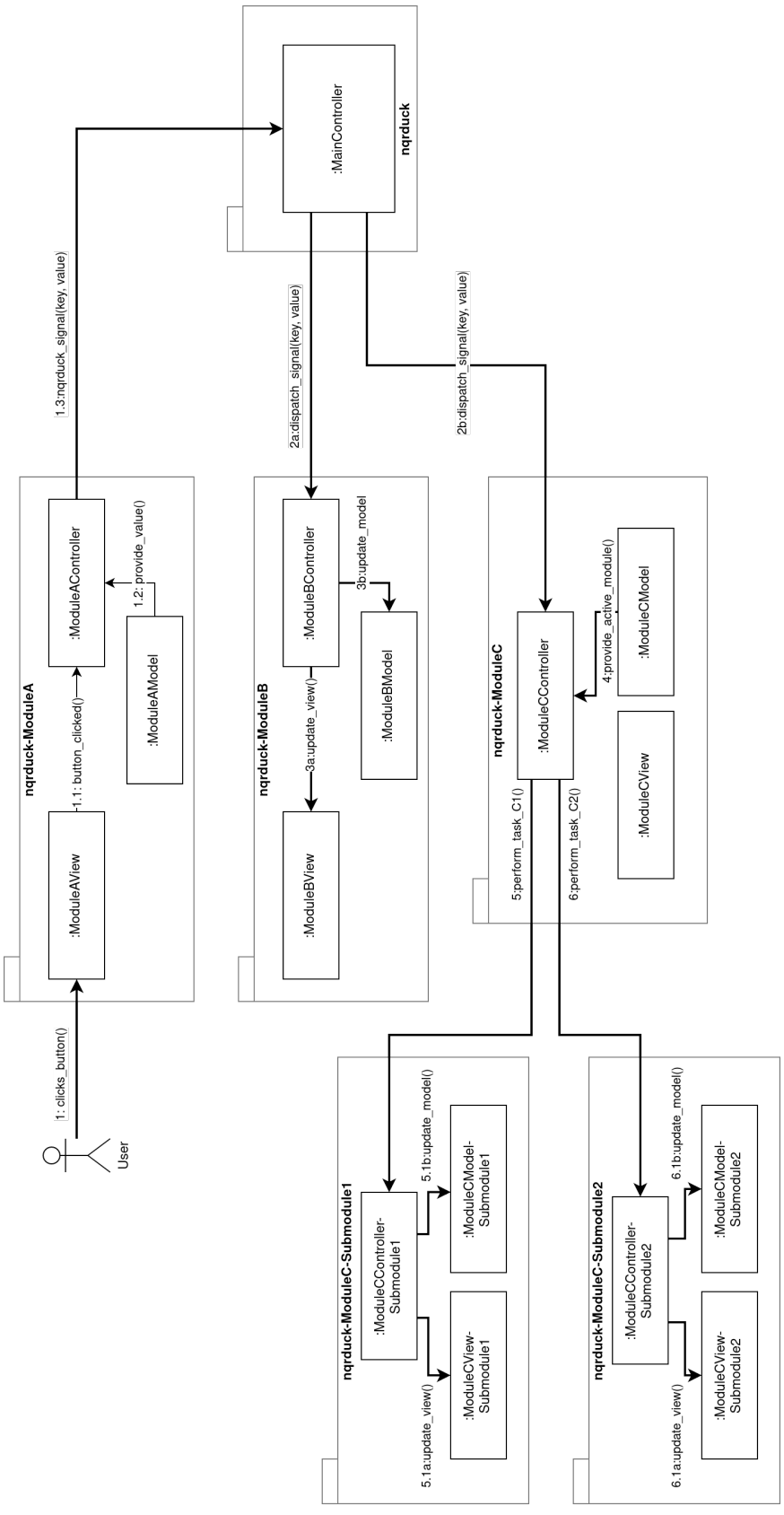


Figure 2.39:: Diagram representing the communication structure of the nqrduck framework.

Figure 2.39 shows an exemplary communication flow within the nqrduck program. It is initiated when the user clicks a button in the GUI:

1. `"clicks_button()"` in the `"ModuleAView"`, initiates the communication process. This triggers a series of method calls within Module A, starting with:
   1.1 `"button_clicked()"` directed to `"ModuleAController"`.
   1.2 `"ModuleAController"` in turn requests a value from the `"ModuleAModel"`: `"provide_value()"`.
   1.3 `"ModuleAController"` then sends a signal with a key-value pair: `"nqrduck_signal(key, value)"` to `"MainController"` within the core module, `"nqrduck"`.
2. `"MainController"` dispatches this signal to two other modules, as indicated by the messages:
   2a `"dispatch_signal(key, value)"` for Module B and
   2b `"dispatch_signal(key, value)"` for Module C.

   The next step depends on the recipient of the signal:
   - If Module B is the intended recipient of the signal (path 3), `"ModuleBController"` then updates:
     3a the view `"update_view()"` and
     3b the model `"update_model()"`.
   - If Module C is targeted (path 4):
     4 `"ModuleCModel"` provides an active module: `"provide_active_module()"`.
     – Depending on which submodule is currently active:
       5 `"perform_task_C1()"` for `"nqrduck-ModuleC-Submodule1"` or
       6 `"perform_task_C2()"` for `"nqrduck-ModuleC-Submodule2"` will be executed.

     These submodules of Module C carry out their actions through messages:
     a. `"update_view()"` and
     b. `"update_model()"`

     contingent upon the active submodule.

2.5. NQRduck Modules

The following sections will describe the different NQRduck modules developed in the course of this project.

2.5.1. Tuning and Matching Module (nqrduck-autotm)

The Tuning and Matching Module is used as a GUI for the ATMV2 described earlier. It is implemented to be loaded into the GUI as a separate tab. The general functionality is described in figure 2.40.

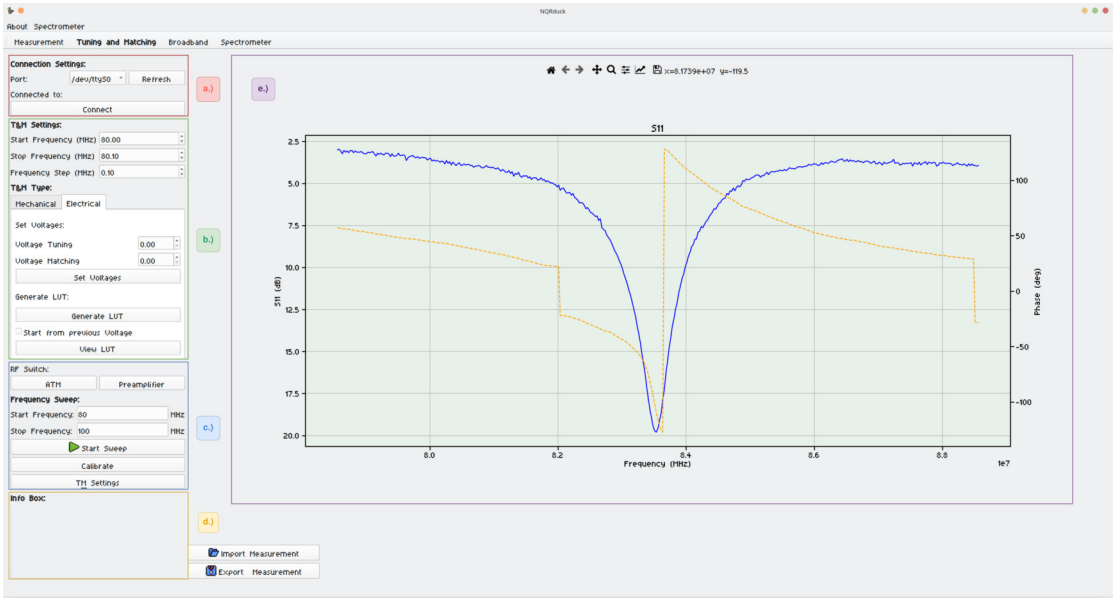


Figure 2.40.: UI of the Tuning and Matching module with highlighted areas: (a) Red area for opening the serial connection to the ATMV2. (b) Green area for switching between tuning and matching in electrically and mechanically tuned probe coils. (c) Blue area for performing frequency sweeps and calibrating the ATMV2 system. (d) Yellow area shows information about the measurement and connection status. (e) Purple area displaying the S_{11} measurement data, with frequency on the x-axis and S_{11} value on the y-axis.

Additionally the module can be used to import and export the S_{11} measurements. For mechanically tuned probe coils, the module can be used to home the steppers or drive them for a relative number of steps or to an absolute position. For the steppers, different positions can be saved in the module to allow the steppers to drive to a previously saved position. For electrically tuned probe coils, the module can be used to set a certain voltage for tuning and matching respectively. Functionality for switching between the signal path for reflection measurements and magnetic resonance experiments is also provided. The parameters for the LUT for electrically and mechanically tuned probe coils

can be set from within the module (start/stop frequency, frequency step). The LUT can then also be viewed and tested from within the GUI.

Communication As described in section 2.3.4, the communication between the ESP32 and the Python Tuning and Matching module takes place via serial communication. The Python module can also receive data from the ATMV2 system. The data format follows the structure of a single character identifying what kind of data is being transmitted, followed by the data itself. How this data is interpreted also depends on the internal state of the module. The different states are described in table 2.7. The data that is received can be found in section 2.3.4, where it corresponds to the output of the different commands.

Implementation Calibration The calibration process is implemented in the *nqrduck-autotm* module. The implementation is based on the One-Port Error model described in section 2.2.2.

1. Define a start frequency, a stop frequency, and a specific number of frequency steps. The frequency resolution is then computed from the number of frequency steps, along with the start and stop frequencies.
2. Terminate the transmission line at the reference plane using the *short* adapter.
3. Measure the reflection coefficient from the start frequency to the stop frequency using the previously calculated frequency steps. The measured reflection coefficient Γ_M is recorded at each individual frequency.
4. Repeat this process with the transmission line terminated with the open adapter and the 50Ω reference load.
5. The system solves the linear equations (2.14) derived from the three different measurements for the error terms at every frequency.

With the completion of the calibration process, the actual reflection coefficient of a DUT at the reference plane can now be measured. This is achieved by solving equation 2.13 at every frequency for which the calibration has been performed. The calibration can then be saved and loaded at a later time for further measurements.

Table 2.7.: Summary of communication logic based on the received data prefix.

Prefix	**Condition**	**Action Performed**
'f'	Frequency sweep has been started	Extracts frequency, return loss, and phase from the received data and adds the data point to the model.
'r'	No active calibration	Indicates measurement completion, stores S_{11} data, hides spinner, and displays frequency sweep duration.
'r'	Active calibration is "short"	Indicates short calibration completion, stores S_{11} data, and resets calibration status.
'r'	Active calibration is "open"	Indicates open calibration completion, stores S_{11} data, and resets calibration status.
'r'	Active calibration is "load"	Indicates load calibration completion, stores S_{11} data, and resets calibration status.
'i'	-	Prepends "ATM Info:" to the received message and displays it in the UI.
'e'	-	Prepends "ATM Error:" to the received message and displays it in the UI.
'v'	LUT is incomplete	Processes voltage sweep results, updates LUT, and triggers the next voltage sweep if necessary.
'v'	-	Update voltages.
'p'	-	Updates the position data for the tuning and matching steppers in the model and marks them as homed. Triggers UI update for active stepper change.
'm'	-	Processes the return loss and phase data from the received message, updates the last reflection in the model with these values.
'z'	-	Processes the positioning data for tuning and matching steppers including the last movement direction, updates their positions and movement direction in the model, adds the data to the mechanical LUT, and determines whether to continue or finish the position sweep.
'c'	-	Sets the signal path to the provided path (pre-amplifier or ATMV2).

2.5.2. Spectrometer Module (nqrduck-spectrometer)

The spectrometer module provides a unified interface for different modules to communicate with spectrometers. The actual spectrometers are then implemented as submodules of this spectrometer module. The spectrometer module provides base classes that every submodule should inherit. These provide a blueprint structure for the spectrometer submodules and implement certain functionality like automatic generation of settings views that can be re-used by the submodules.

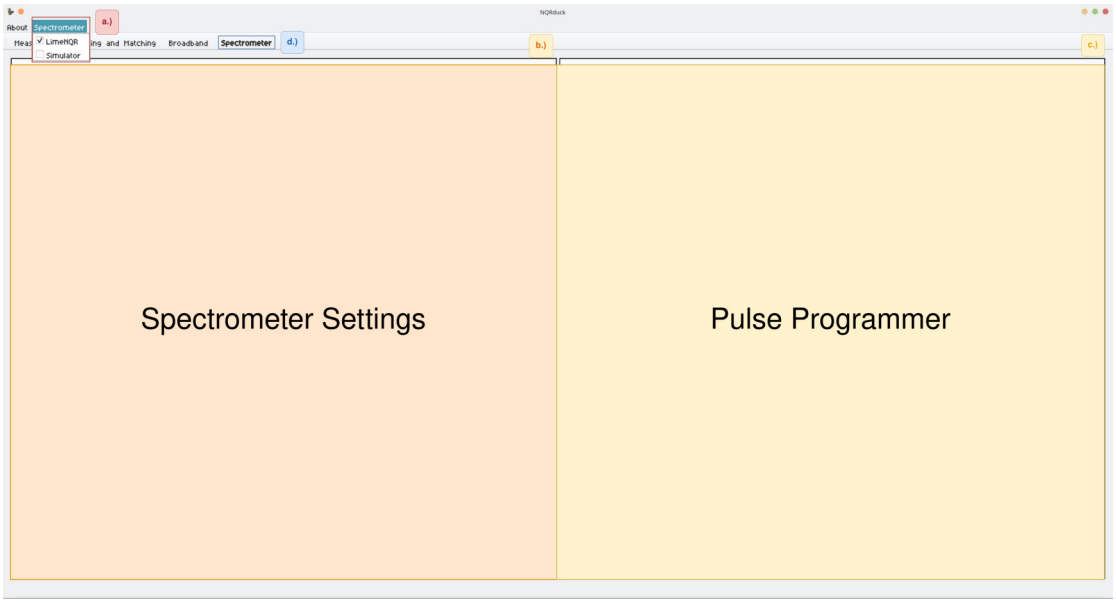


Figure 2.41.: Spectrometer Module with different areas highlighted. The active spectrometer-submodule is for the LimeNQR spectrometer. (a) Red area is used to switch between different spectrometer submodules. (b) Orange area is the different settings of the active spectrometer-submodule. For this image the section was marked ('Spectrometer Settings') to give a general understanding of the GUI structure (c) Yellow area indicates the pulse programmer of the currently active spectrometer. The area was marked 'Pulse Programmer'. (d) In the blue area it can be seen that the spectrometer module is currently active in our NQRduck main window.

The module itself has the MVC architecture of all NQRduck modules as can be seen in figure 2.42.

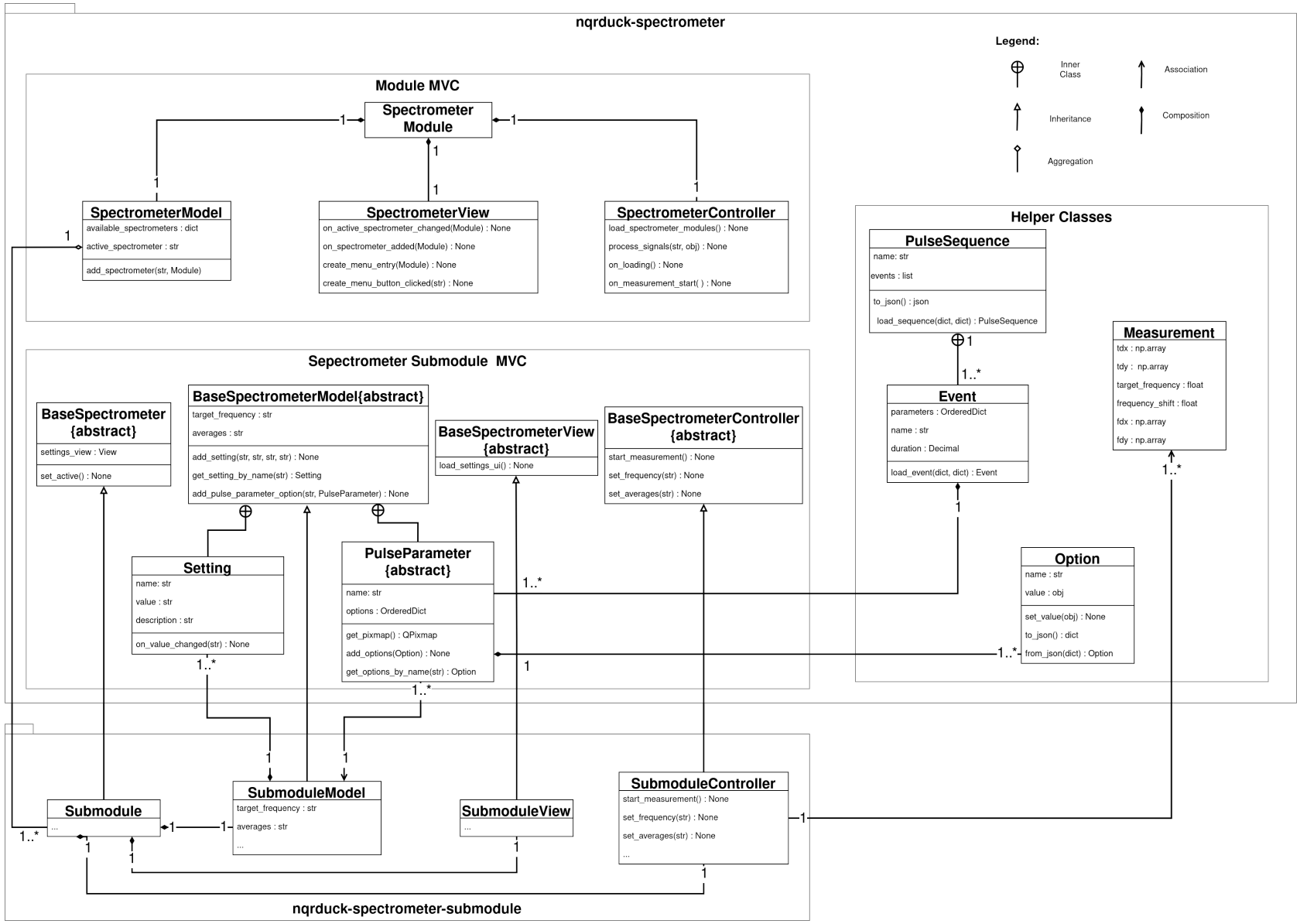


Figure 2.42.: Simplified class diagram of the spectrometer module. The exact structure of the *PulseSequence* was omitted for simplicity and can be found in figure 2.46.

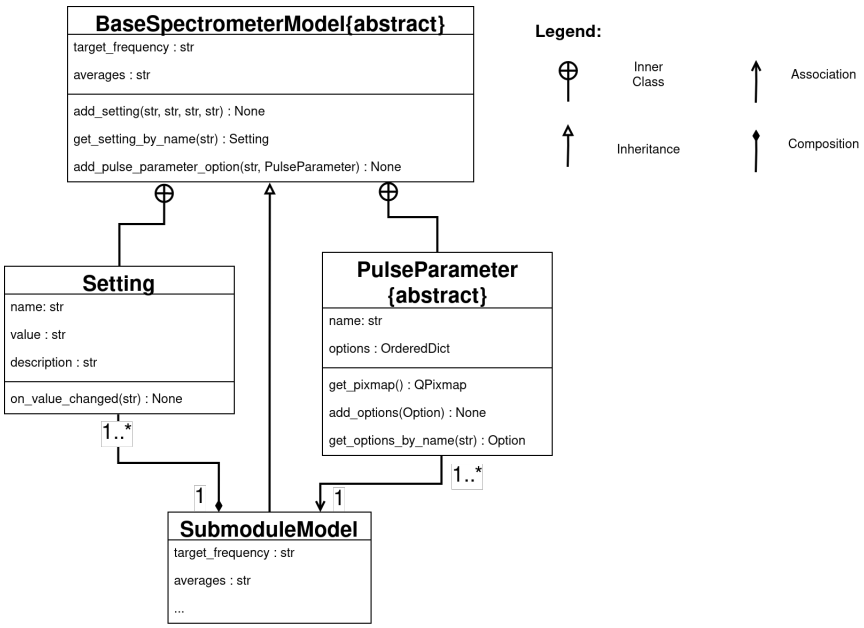


Figure 2.43.: Class diagram zoomed in on the composition of the spectrometer model.

The spectrometer module itself is not used for the communication with the actual spectrometer hardware, this implementation is done in the spectrometer submodules. The spectrometer class however provides a unified interface for communication with different spectrometer submodules. As an example, every spectrometer will have the option to start a measurement. When the spectrometer module receives a command to start a measurement, it delegates the task to the appropriate submodule, which then converts the command into instructions for the hardware. As figure 2.42 shows, all submodules of the spectrometer module have classes inheriting from the *BaseSpectrometer* classes.

Figure 2.42 shows also the basic composition of every spectrometer submodule implementation. As defined in previous chapters, the Model describes our spectrometer composition (Figure 2.43). This means in the module we define two different things:

1. Settings with a value, name and a description. These settings are the same for every event in our pulse sequence. One example would be the setting for the ADC sampling frequency of the experiment.
2. Different **Classes** of *PulseParameters*. These *PulseParameters* will be instantiated for every event of our pulse sequence and can therefore hold different values for each event. One example would be the 'RX' *PulseParameter* which can be either True or False for the different events of our *PulseSequence*.

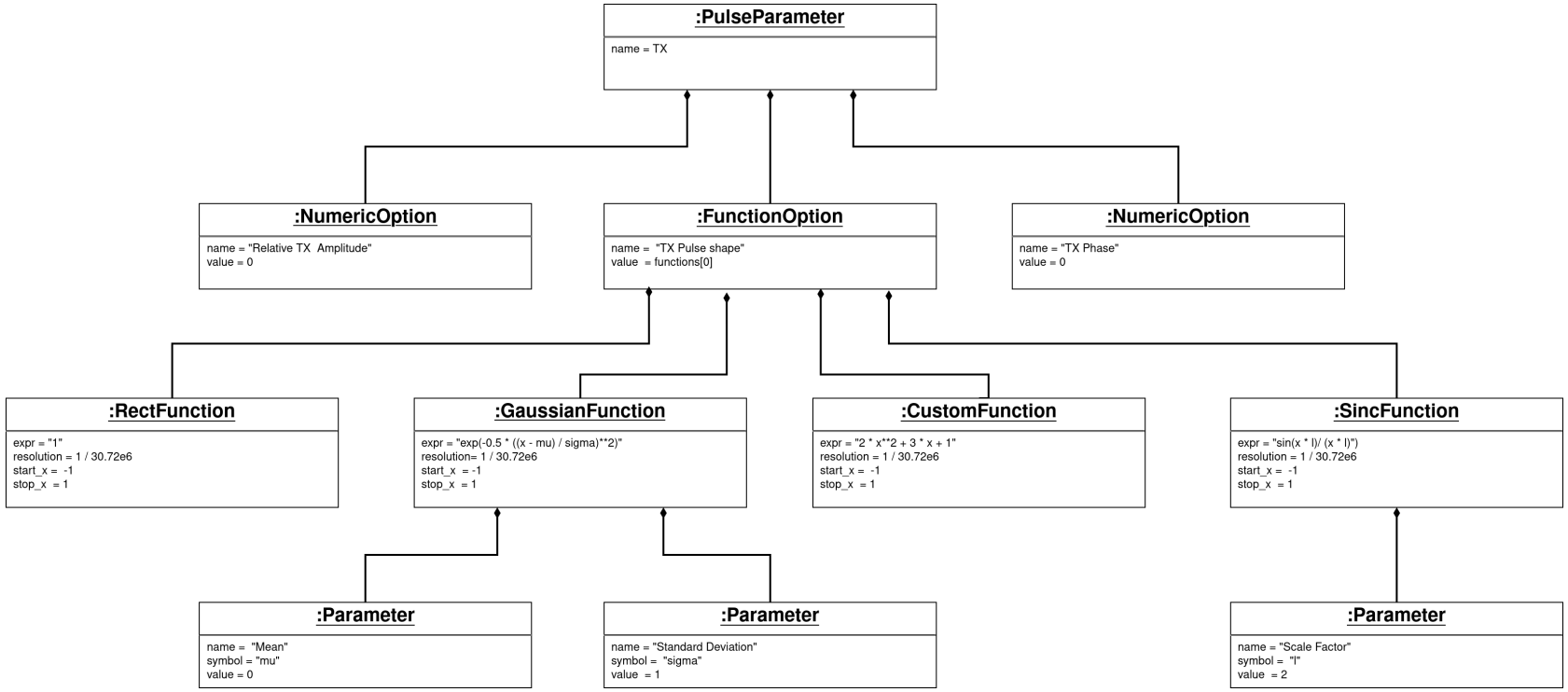


Figure 2.44.: Diagram of the different objects a TX pulse is composed of. It can be seen that a *TXPulse* can have different options: Two *NumericOptions* for the relative amplitude and the phase of the pulse. Furthermore there is one *FunctionOption* for the pulse shape. Here there are 4 different *Function* options (*RectFunction*. *GaussFunction*, *SincFunction* and *CustomFunction*). Some of the *Functions* have associated *Parameters* that are assessed when evaluating the *Function* expression.

Pulse Parameters In figure 2.44, the exemplary composition of a *TXPulse* is depicted. Different options must be added in accordance with the *PulseParameter* that is to be created, if an own *PulseParameter* is desired. This *PulseParameter* would then be added to the according spectrometer submodule model.

View Generation As can be seen in figure 2.41, a spectrometer can have different settings. The GUI for this settings can be automatically generated through the *BaseSpectromterView* class using the *load_settings_ui()* method. This significantly speeds up implementation of new spectrometers, since it only has to be described what kind of settings a spectrometer has and the GUI is then automatically generated.

Figure 2.45.: Automatically generated settings view for the LimeNQR spectrometer. The view was cropped to a number of settings. This is then added to the orange 'Spectrometer Settings' region in figure 2.41.

Pulse Sequence The spectrometer module also provides a class that describes how a pulse sequence looks. Spectrometer submodules are not required to follow this structure, but it can be used to make integration of additional spectrometers easier. The class diagram of the pulse sequence is depicted in figure 2.46.

Notes It should be noted that the presented structure can be seen as a suggestion for how one might implement a spectrometer submodule. However, this is not enforced in the code. The only determinant if a module is loaded as a submodule of the spectrometer module is if the Module itself inherits from the *BaseSpectrometer* class.

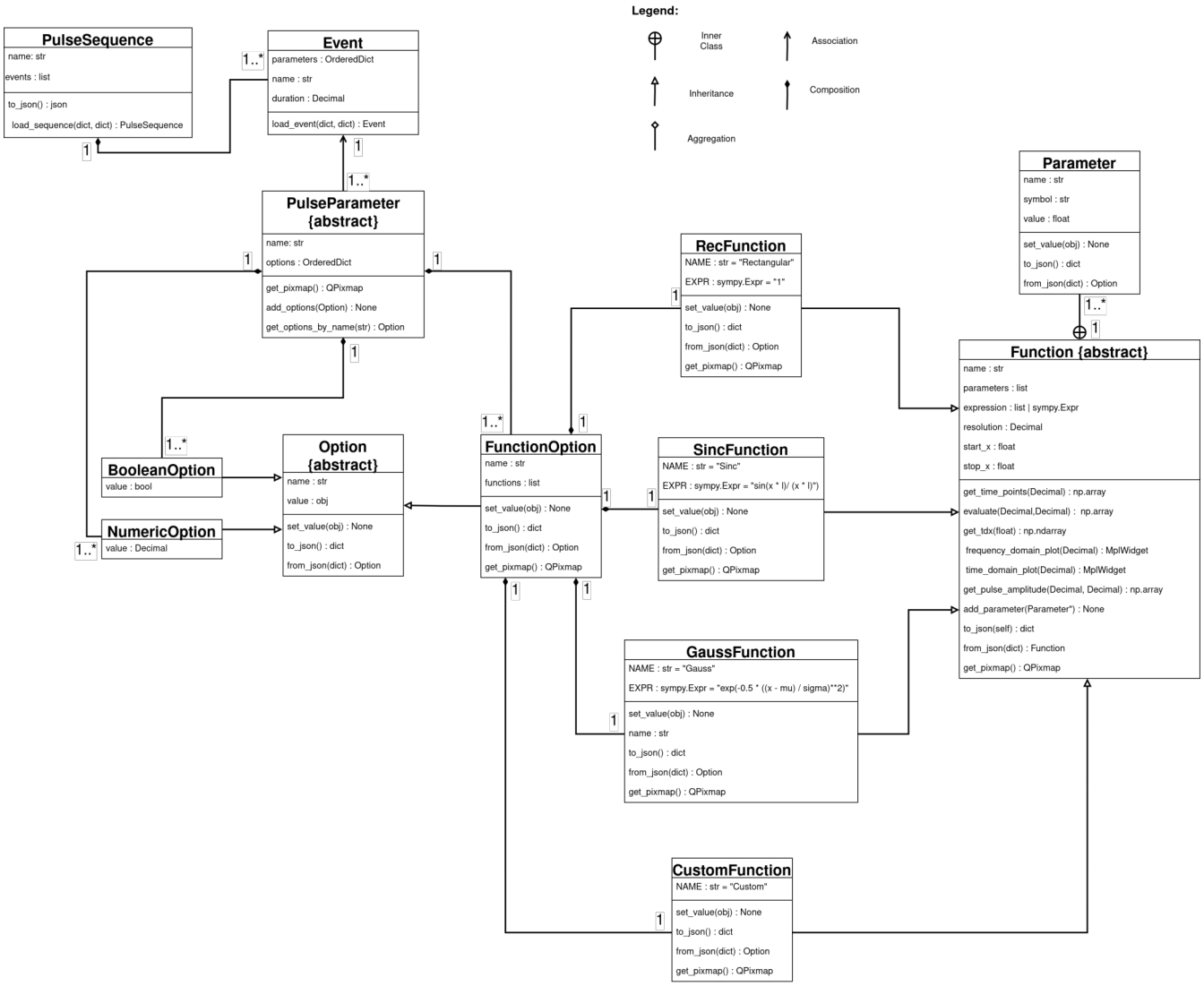


Figure 2.46.: Class diagram on the composition of a pulse sequence. A pulse sequence consists of multiple different *Events* that are executed in sequence. Every *Event* has a number of *PulseParameters* for the spectrometer (e.g. RX and TX). These *PulseParameters* have different *Options* associated with them. For example, the RX *PulseParameter* has a *BooleanOption* that can be either True or False. A more complex PulseParameter would be the TX *PulseParameter* which can have Functions associated with it for different pulse shapes.

2.5.3. LimeNQR Spectrometer Module (nqrduck-spectrometer-limenqr)

This module is a submodule to the spectrometer module described in the previous section (2.5.2).

This module has two primary tasks:

1. It describes the structure of the spectrometer: What settings the spectrometer has and what pulse parameters can be set for a pulse sequence. The settings can be found in table 2.8.
2. When a measurement is started, this module will translate the pulse sequence to instructions that can be used with the spectrometer.

For communication with the LimeNQR spectrometer, the module relies on Python and C++ programs provided by Doll [5] as mentioned in section 2.3.1. The communication flow of the module is depicted in figure 2.47.

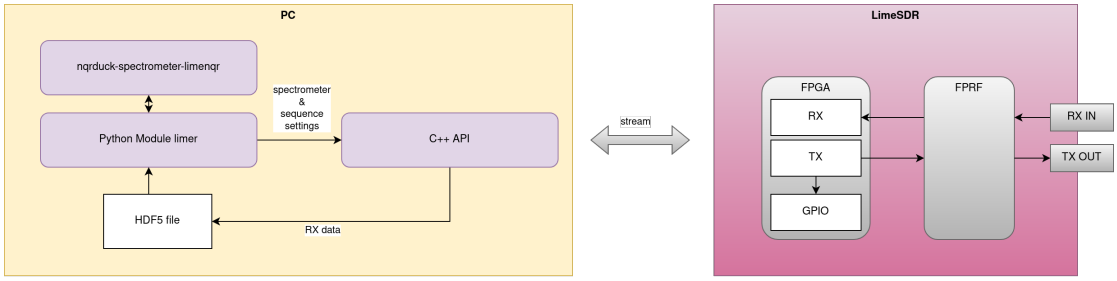


Figure 2.47.: Diagram depicting the communication of the different components used for the LimeNQR spectrometer. Figure adapted from [5].

Whenever a measurement is started, the *start_measurement* signal will be dispatched to the LimeNQR spectrometer module. The module itself will then translate the associated *PulseSequence* object by setting specific attributes of the *limr* module. This can for example be the LO frequency or timing of TX pulses.

The module itself provides a GUI as depicted in 2.48. Its *PulseParameters* are the 'TX' parameter described in figure 2.44 and the 'RX' option determining at which point in the pulse sequence the experiment data should be read. Furthermore, the settings of the spectrometer can changed via the GUI. The different available settings are listed in 2.8.

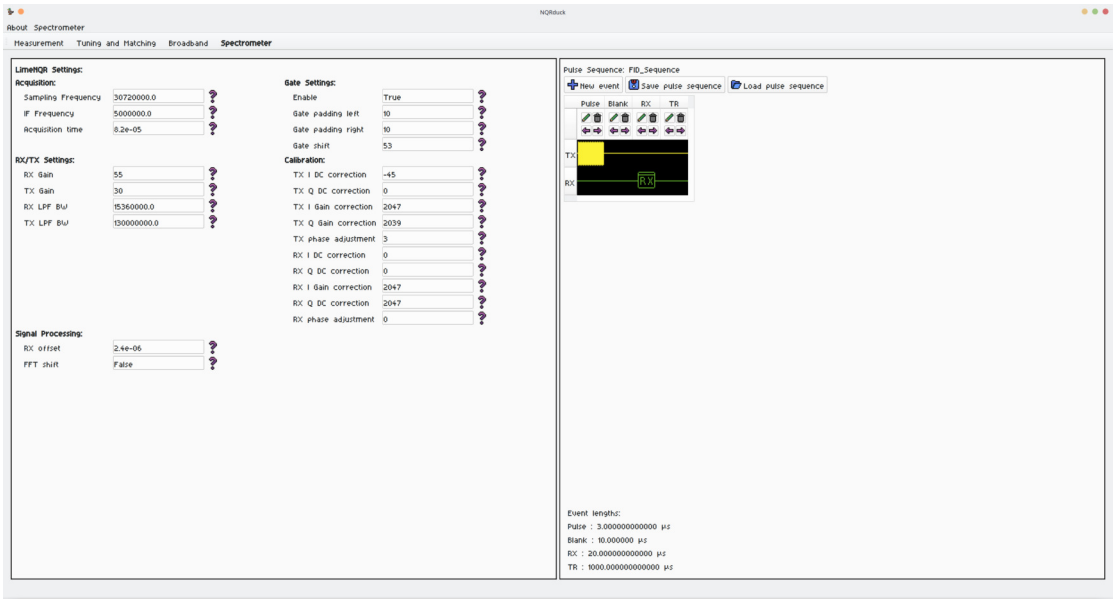


Figure 2.48.: UI of the LimeNQR spectrometer. On the left side the settings can be seen, and on the right side the pulse programmer. An exemplary FID sequence was programmed to showcase the look of the module. The settings view was automatically generated using the *generate_settings_view()* described in section 2.5.2. The pulse programmer is the module described in 2.5.6. This means all of the depicted GUI is automatically generated based on the LimeNQR spectrometer model.

Table 2.8.: Settings for the LimeNQR Spectrometer Module.

Setting	Description
Acquisition Settings	
Sampling Frequency	The rate at which the spectrometer samples the input signal.
IF Frequency	The intermediate frequency to which the input signal is downconverted during analog-to-digital conversion.
Acquisition Time	The interval after the pulse sequence starts in which the spectrometer begins acquiring data.
Gate Settings	
Gate Enable	A setting that controls whether gating is on during transmitting.
Gate Padding Left	The number of samples by which to extend the gate window to the left.
Gate Padding Right	The number of samples by which to extend the gate window to the right.
Gate Shift	The delay, in number of samples, by which the gate window is shifted.
RX/TX Settings	
RX Gain	The gain level of the receiver's amplifier.
TX Gain	The gain level of the transmitter's amplifier.
RX LPF BW	The bandwidth of the receiver's low-pass filter which attenuates frequencies below a certain threshold.
TX LPF BW	The bandwidth of the transmitter's low-pass filter which limits the frequency range of the transmitted signal.
Calibration Settings	
TX I/Q DC Correction	Adjusts the direct current offset errors in the in-phase (I) and quadrature (Q) components of the transmit (TX) path.
TX I/Q Gain Correction	Modifies the gain settings for the I and Q channels of the TX path, adjusting for imbalances.
TX Phase Adjustment	Corrects the phase of the TX signal to ensure proper timing and synchronization.
RX I/Q DC Correction	Adjusts the direct current offset errors in the in-phase (I) and quadrature (Q) components of the receive (RX) path.
RX I/Q Gain Correction	Modifies the gain settings for the I and Q channels of the RX path, correcting for imbalances.
RX Phase Adjustment	Calibrates the phase of the RX signal to match signal processing requirements.
Signal Processing Settings	
RX Offset	The time delay before the RX event is processed in microseconds.
FFT Shift	A Boolean setting that controls whether an FFT shift operation is applied to the signal processing routine.

2.5.4. Simulator Spectrometer Module (nqrduck-spectrometer-blochsimulator)

The Simulator module is used to simulate magnetic resonance experiments. It is a direct translation of the code provided by Graf from *MATLAB* to Python [43] [44]. Furthermore, it is based on an unpublished simulator tool developed by Nißl and Scharfetter at the Institute of Biomedical Imaging (TU Graz). In this simulation, the excitation field was calculated from various hardware specifications like coil geometry and PA power. The NQR signal is then calculated from a reference voltage and the magnetization simulated by the Bloch simulator. A description of this can be found in the Appendix C.2.

The simulator is implemented as a submodule of the spectrometer module. This allows reusing of previously explained features of the nqrduck-spectrometer module. However, instead of translating the associated *PulseSequence* object to hardware instructions like in the LimeNQR spectrometer module, the pulse sequence is translated to input to a Bloch simulator.

Since this module has exactly the same *PulseParameters* as the LimeNQR spectrometer module, the programmed pulse sequences can be used by both modules interchangeably.

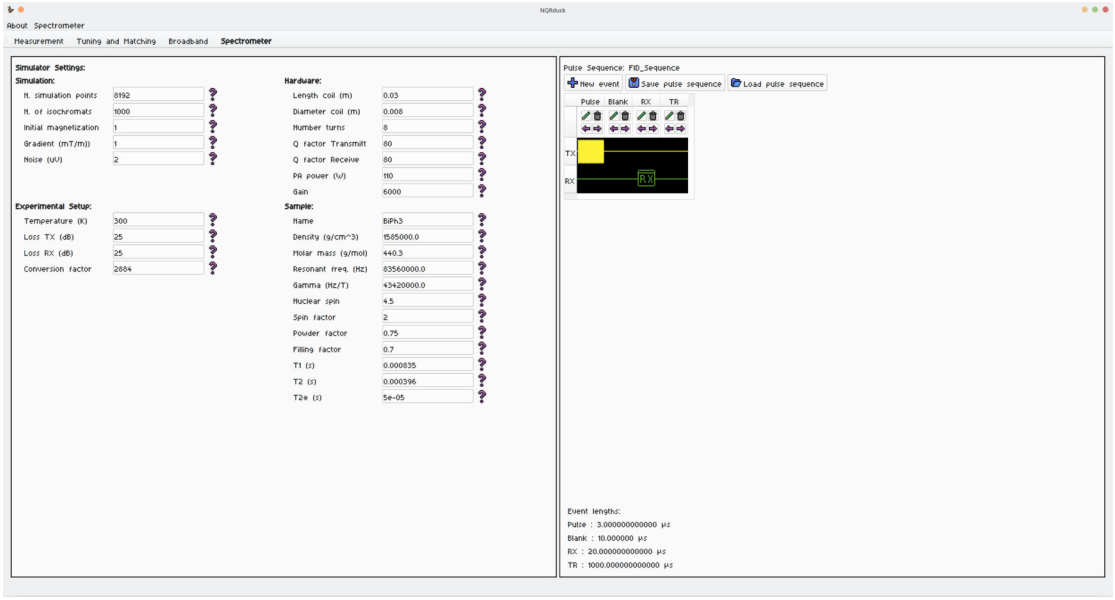


Figure 2.49.: UI of the simulator spectrometer module. On the left side the settings can be seen, and on the right side the pulse programmer. The similarity to figure 2.48 should be noted. This is because both spectrometers have the same *PulseParameters* and only different settings.

Tables 2.9 & 2.10 depict the different settings for the simulator.

Table 2.9.: Settings for the Simulator Module (Part 1). Descriptions are provided to detail the function of each parameter within the simulation.

Setting	Description
Simulation Settings	
N. simulation points	The total points in the discrete representation of the simulated signal.
N. of isochromats	Defines the number of discrete subsections of the sample with independently evolving magnetizations.
Initial magnetization	Sets the starting magnetization value for the simulation, before any RF pulses are applied.
Noise (uV)	Adds a specified level of random noise to the simulation to mimic real-world signal variations.
Hardware Settings	
Length coil (m)	The length of the coil within the hardware setup.
Diameter coil (m)	The diameter of the coil.
Number turns	Total turns of wire in the coil.
Q Factor transmit	The quality factor of the transmit path, which has an effect on the field strength for excitation.
Q Factor receive	The quality factor of the receive path, which has an effect on the final SNR.
PA power (W)	The power output capability of the power amplifier, determines the strength of pulses that can be generated.
Gain	The amplification factor of the receiver chain, impacting the final measured signal amplitude.

Table 2.10.: Settings for the Simulator Module (Part 2). Descriptions are provided to detail the function of each parameter within the simulation.

Experimental Setup Settings	
Temperature (K)	The absolute temperature during the experiment, which can influence the signal-to-noise ratio.
Loss TX (dB)	The signal loss occurring in the transmission path, affecting the effective RF pulse power.
Loss RX (dB)	The signal loss in the reception path, which can reduce the signal that is ultimately detected.
Conversion Factor	The factor to convert from Volts to the spectrometer units.
Sample Settings	
Name	The name or identifier of the sample being simulated.
Density (g/cm^3)	The density of the sample, utilized in the calculation of the initial signal.
Molar mass (g/mol)	The molar mass of the substance being analyzed.
Resonant freq. (Hz)	The resonant frequency of the sample.
Gamma (Hz/T)	The gyromagnetic ratio of the sample's nuclei.
Nuclear spin	The intrinsic spin number of the sample's nuclei.
Spin factor	The spin factor represents the scaling coefficient for observable nuclear spin transitions along the x-axis, derived from the Pauli I_x^0-matrix elements.
Powder factor	A factor representing the crystallinity of the solid sample.
Filling factor	The ratio of the sample volume that occupies the coil's sensitive volume.
T1 (s)	The longitudinal or spin-lattice relaxation time of the sample, influencing signal recovery between pulses.
T2 (s)	The transverse or spin-spin relaxation time, determining the rate at which spins dephase and the signal decays in the xy plane.
T2* (s)	The effective transverse relaxation time, incorporating effects of EFG inhomogeneities and other dephasing factors.

2.5.5. SCOUT Spectrometer Module (nqrduck-spectrometer-scout)

The SCOUT [10] (*Tecmag, Houston, U.S.*) spectrometer is a commercial spectrometer used for research at the Institute of Biomedical Imaging (TU Graz). It is primarily used for broadband NQR measurements of samples with unknown resonance frequencies [25]. Currently the SCOUT spectrometer is located at University of Graz (Institute of Physics), where it was used for NQR measurements of Uranium-235-doped Calcium fluorite crystals in liquid helium.

The manufacturer provides software with the spectrometer which is used for pulse sequence programming and data acquisition (TNMR, *Tecmag, Houston, U.S.*). However, the software does not include functionality for broadband measurements and automatic Tuning and Matching of different probe coils. Previously this functionality was provided by the ScoutApp, a MATLAB (*MathWorks, Natick, U.S.*) program, implemented by Wunderl [45].

To allow use of the SCOUT spectrometer within the NQRduck framework, especially in the context of broadband measurements, a spectrometer submodule for control of the spectrometer was implemented. For this purpose, the Object Linking and Embedding (OLE) functionality provided by the TNMR program is used. This provides an interface for spectrometer control and data processing for external applications.

Using the OLE interface, the NQRduck module can change spectrometer settings and start measurements. Additionally, it can request the measurement data. This data can then be processed further within the NQRduck framework.

For Tuning and Matching of the different probe coils, the NQRduck module for automatic Tuning and Matching can be used. The NQRduck coordinates the tasks of the different modules and hardware. For broadband measurements the probe coil is first tuned and matched using the nqrduck-autotm module. Then the measurement is conducted using the SCOUT spectrometer module. The measurement data is then further processed by the nqrduck-broadband module.

Pulse sequences for the SCOUT spectrometer are still programmed within the TNMR program. A button to open the TNMR program is displayed within the SCOUT Spectrometer module.

The TNMR program is only available for Windows (*Microsoft, Redmond, U.S.*) operating systems. Therefore, this module can only be used with Windows.

[10] https://tecmag.com/scout/

2.5.6. Pulse Programmer Module (nqrduck-pulseprogrammer)

The pulse programmer module is an event-based graphical representation of a pulse sequence. The main idea is to have a variety of pulse parameter options like 'Receive' and 'Transmit', where different events can be created that are then executed in sequence. The pulse programmer module relies on the *PulseSequence* structure discussed in Figure 2.46. It primarily serves as a visualization tool for viewing and programming of said structure.

The pulse programmer module can also be used to load and save pulse sequence files. These files have the extension *.quack* and are structured as JavaScript Object Notation (JSON) files. An example can be found in the appendix C.3.

The pulse sequence is represented as a table. In the columns are the different events that are executed in sequence. In the rows are the different pulse parameter options of the spectrometer (Figure 2.50).

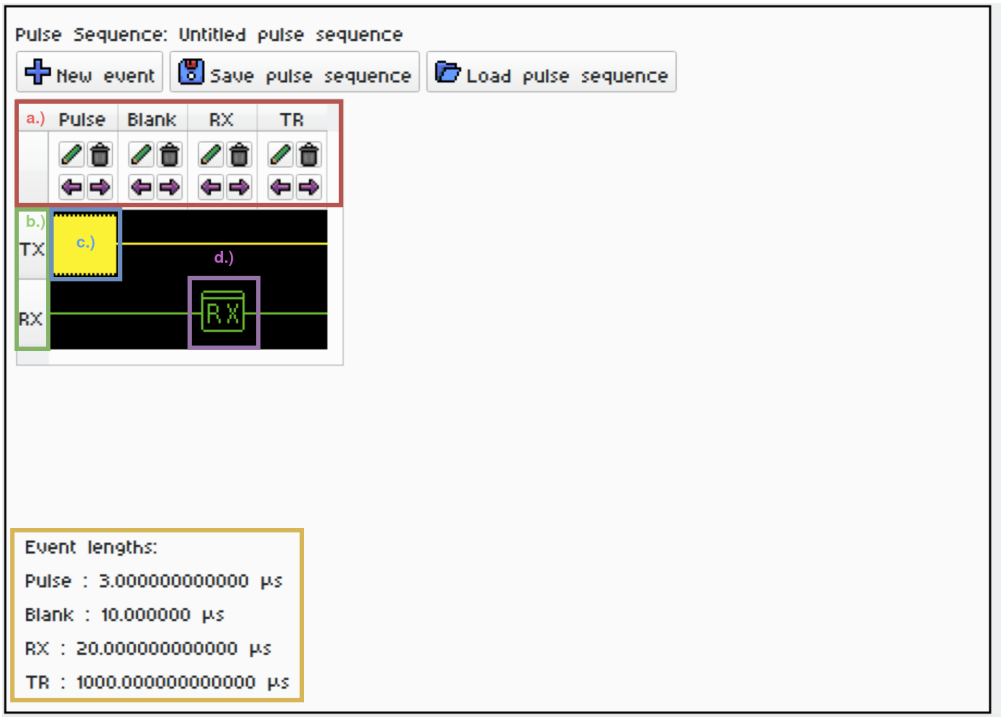


Figure 2.50.: Screenshot of the pulse programmer in the NQRduck showcasing an FID sequence for the LimeNQR spectrometer. This sequence features four events: 'Pulse', 'Blank', 'RX', and 'TR' (red box a.). The spectrometer's two pulse parameter options are shown in green box b.) ('TX' and 'RX'). 'Pulse' refers to a $3\mu s$ rectangular transmit pulse via the TX port, modifiable by the 'TX' parameter (blue box c.). 'Blank' denotes a $10\mu s$ period for coil-ringdown. 'RX', at $20\mu s$, is the ADC read-out phase (violet box d.), which can be toggled on or off. Lastly, 'TR' is the repetition time, spanning $1000\mu s$.

The pulse parameter options depend on the spectrometer the pulse program-

mer module is associated with. For example, the *LimeNQR* spectrometer has the pulse parameter options 'TX' and 'RX'. The pulse programmer module now adds the according rows to its table. Every pulse parameter option then has a certain value for every entry in the table. Depending on the state of the different *PulseParameter* objects they will provide different *QPixmap* objects. As an example, the *PulseParameter* option 'TX' with the *NumericOption* for 'Relative TX Amplitude' larger than zero and a *FunctionOption* that is a *RectFunction* will return an image of a rectangular pulse. If the 'Relative TX Amplitude' would be zero however, the image would simply be a yellow line depicting an idle state.

The *Options* associated with a certain event and *PulseParameter* can be modified by clicking onto the according cell. A new window will open where the *Options* can be modified. The view for this is automatically generated. An example for the 'TX' *PulseParameter* described in figure 2.44 can be seen in figure 2.51. The different options are added to the window as rows.

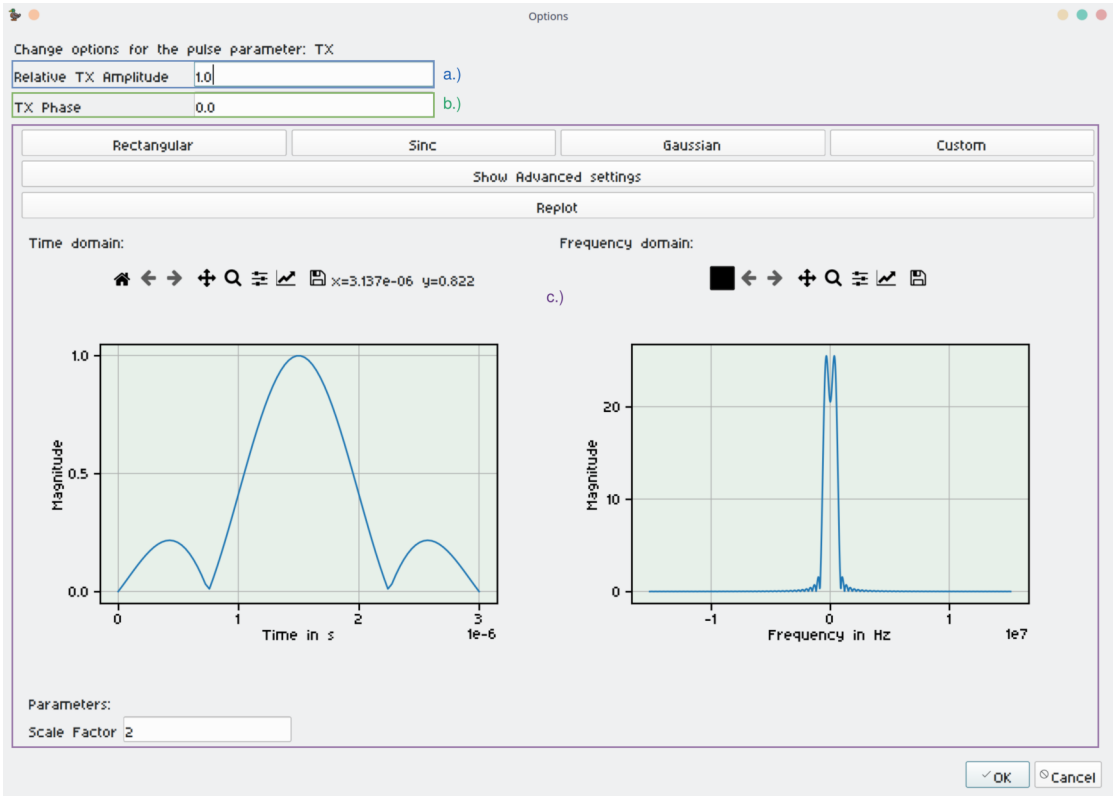


Figure 2.51.: UI of the pulse programmer module when modifying a *PulseParameter*. Different sections of the window are highlighted. (a) Blue is the *NumericOption* for the 'Relative TX Amplitude'. (b) Green is the *NumericOption* for the 'TX Phase'. (c) Violet is the *FunctionOption* with different selectable functions.

2.5.7. Measurement Module (nqrduck-measurement)

The measurement module is used for single frequency magnetic resonance experiments. It serves as a unified GUI for the LimeNQR spectrometer module as well as for the simulator module for single frequency experiments. If new functionality (e.g. signal processing) is implemented once in the measurement module, it is available for every spectrometer that uses the measurement module. The GUI is depicted in figure 2.52.

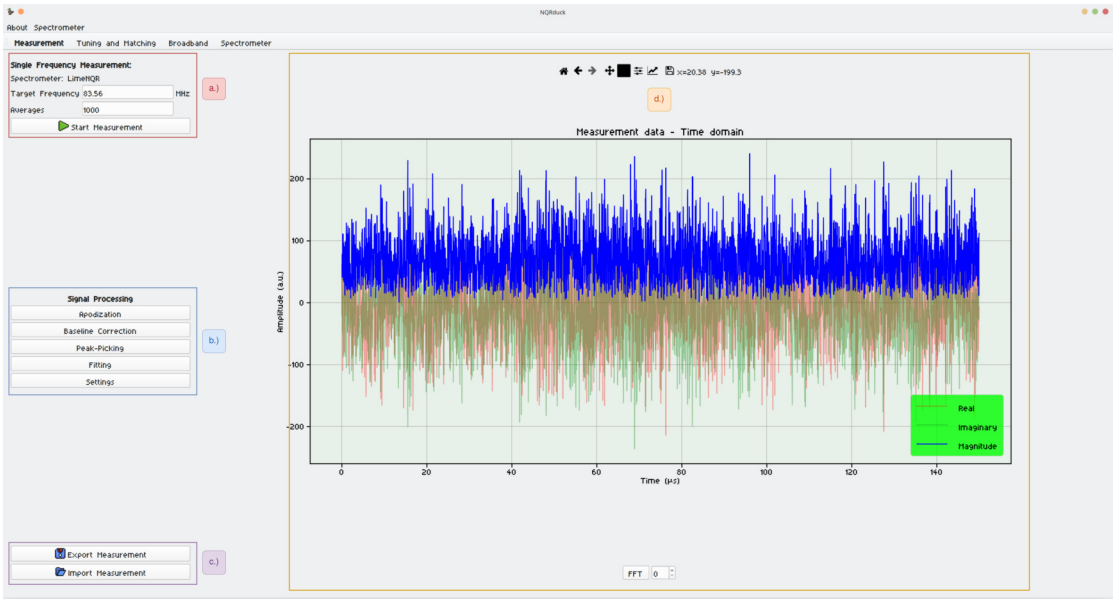


Figure 2.52.: UI of the measurement module with different sections highlighted. (a) Red section shows the basic measurement settings like target frequency for the experiment and number of averages. (b) Blue section shows a number of different signal processing settings. (c) Violet section shows options used for importing and exporting of experimental data. (d) Orange section shows the visualization of the experimental data. The plot can be switched between time-domain and frequency-domain view.

It can be used to perform post-processing on the experimental data. Different post-processing algorithms are:

- Apodization
- Baseline Correction
- Peak Picking
- Fitting

The module also allows for saving and loading of previous experiments.

2.5.8. Broadband Module (nqrduck-broadband)

The broadband module is used for measurements over a larger frequency range. The broadband spectrum is assembled from multiple single-frequency experiments.

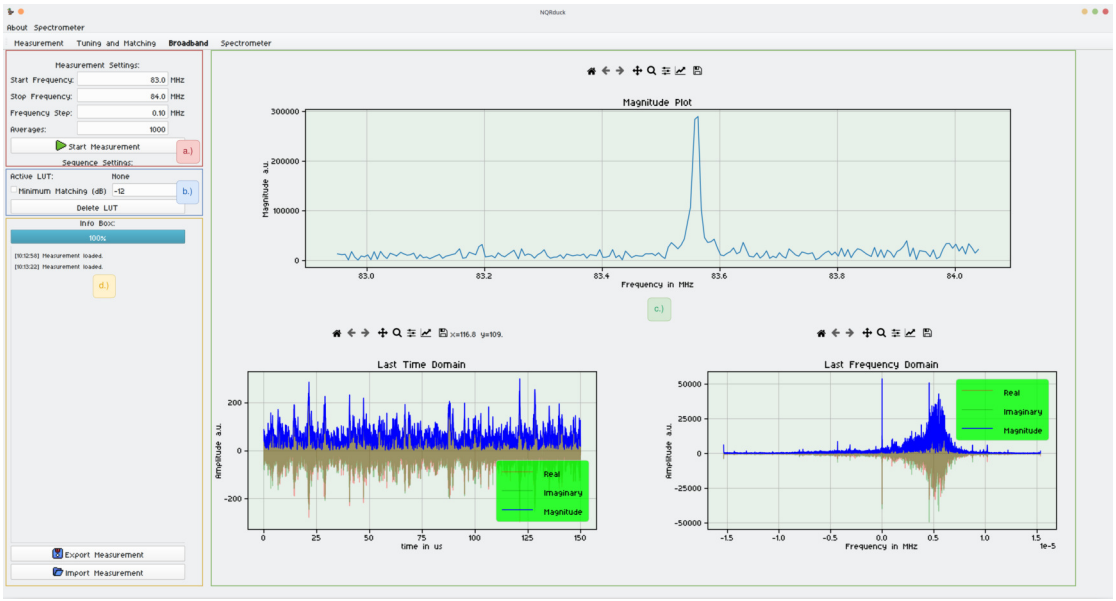


Figure 2.53.: UI of the broadband module with different sections highlighted. (a) Red section are the experiment settings. The start and stop frequency, the frequency step-size and the number of averages can be set here. (b) Blue section is an area containing information about the recently active LUT. (c) Green section is the data about the broadband experiment. In the upper plot, the assembled amplitude spectrum in frequency domain is displayed. In the lower two plots, the time- and frequency-domain data of the last single frequency experiment is displayed. (d) Yellow section is an area containing various information about the broadband measurement from completion status to errors and warnings.

The broadband module can be used with all three different probe coils described in section 2.3.2. For Tuning and Matching of the electrically and mechanically tuned probe coils, the broadband module depends on the NQRduck module for Tuning and Matching (Section 2.5.1). At first a LUT is generated for the appropriate probe coil by the Tuning and Matching module. Then, when the broadband measurement is conducted, this LUT is used for Tuning and Matching the probe coil. For the spectrometer control and pulse sequence programming, the broadband module depends on the spectrometer module described in section 2.5.2.

The broadband measurement process can therefore be described by the following flowchart (Figure 2.54).

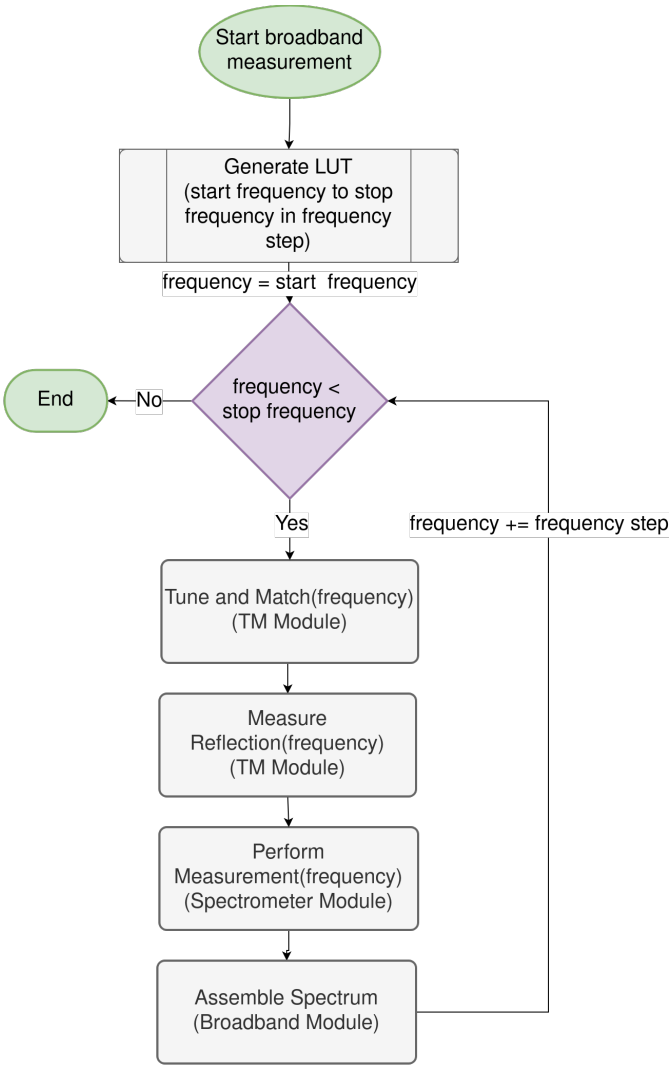


Figure 2.54.: Flowchart of the broadband measurement process. At first, a LUT is generated in the desired frequency range with the specified frequency step size. Then, the probe coil is tuned and matched to the specified frequency. The reflection is then measured at the desired frequency. A measurement is then started using the spectrometer module. The broadband spectrum is now assembled with the new measurement data. The frequency is now incremented by the specified frequency step. This process is repeated until the specified frequency range has been scanned.

The principle of spectrum assembly is depicted in figure 2.55.

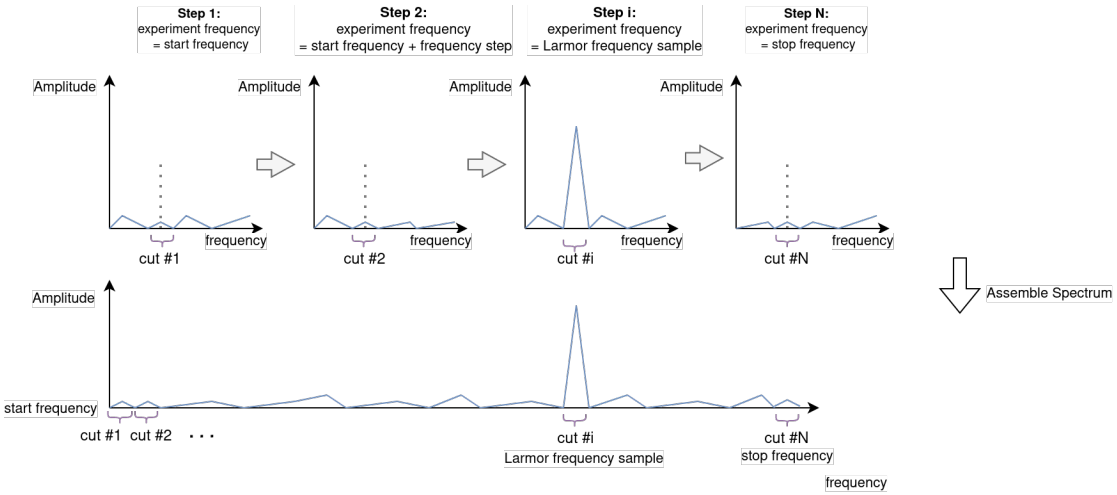


Figure 2.55.: Principle behind spectrum assembly.

The module offers features for loading and saving broadband measurements. Moreover, it records the reflection of each single frequency experiment, including this data into the overall broadband plot. This capability enables users to track the probe coil's matching performance across varying frequencies.

2.6. Experimental Setup

The following sections describe the experimental setup used for this project. The experimental setup aims to showcase the different functionalities implemented for this project. Because the GUI is a significant part of the project, the plots were taken directly from the program whenever possible. If the plots were modified in any way, this will be indicated in the figure caption.

2.6.1. S_{11} Measurements

The S_{11} measurement should demonstrate that the ATMV2 system reliably measures the reflection coefficient of different probe coils at different frequencies.

For the S_{11} measurement, the probe coil was first measured with a Vector Network Analzyer (VNA) ZVL3 (*Rohde & Schwarz, Munich, Germany*) (Figures 2.56 & 2.57). This is seen as the reference for the measurement. The VNA data was plotted using Matplotlib [11]. Then, the same measurements were conducted using the NQRduck in combination with the ATMV2 system.

Mechanically Tuned Probe Coil

The mechanically tuned probe coil was tuned and matched to a 83.56MHz frequency using the VNA. Then, the S_{11} measurement with the NQRduck in combination with the ATMV2 was performed according to figure 2.56. Additionally, the signal levels for the S_{11} measurements using the ATMV2 system were varied by adding and removing a 10dB attenuator to the reflectometer.

The measurement using the ATMV2 was conducted in a frequency range of 78.56MHz to 88.56MHz with 401 points. Additionally a second measurement was performed with a reduced frequency range from 82.56MHz to 84.56 MHz with 401 frequency points.

Electrically Tuned Probe Coil

For electrically tuned probe coils, the custom-built Cryogen-4 (*Institute of Biomedical Imaging, Graz, Austria*) was used [4]. First the probe coil was tuned and matched with output voltages from the ATMV2 system while observing the reflection on the VNA. This measurement was performed to show that the output signal levels of the ATMV2 system do not detune the coil.

Then the S_{11} measurement was performed using the NQRduck in combination with the ATMV2 system (Figure 2.57). This was again performed in the two previously defined frequency ranges with 401 points.

[11] https://matplotlib.org/

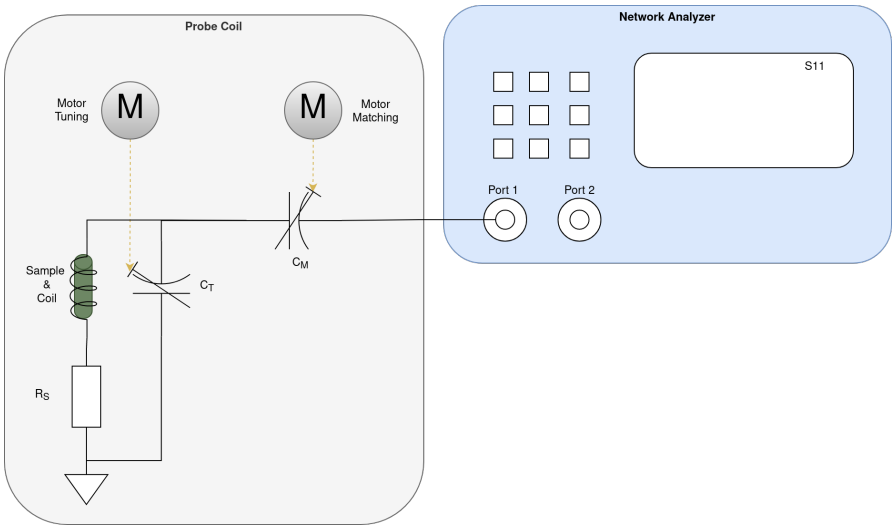


Figure 2.56.: Schematic illustration of the setup used for the S_{11} measurement of mechanically tuned probe coils.

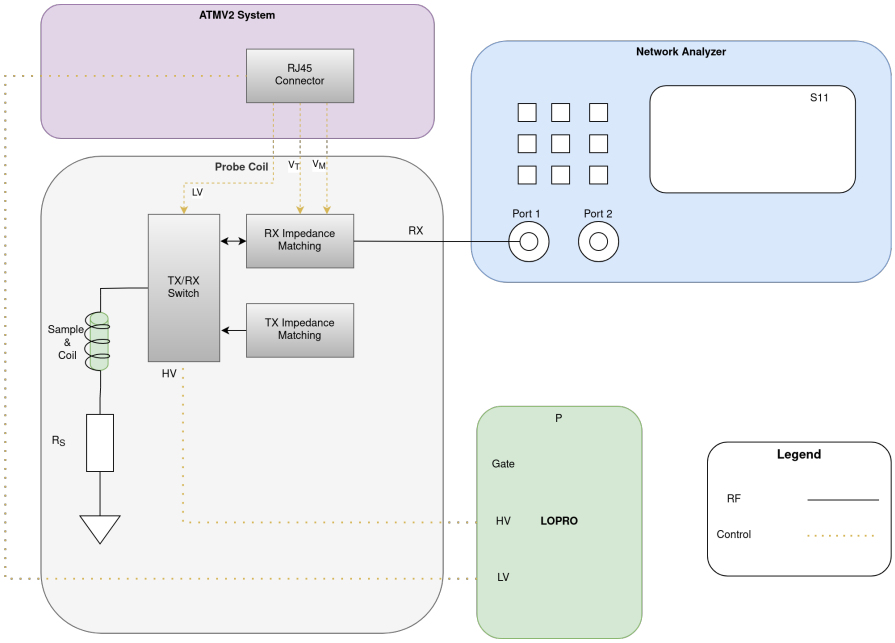


Figure 2.57.: Schematic illustration of the setup used for the S_{11} measurement of electrically tuned probe coils. For the measurement, it is important to note that all of the ground planes of the different devices (VNA, probe coil, ATMV2 and LOPRO) need to be connected properly or the measurement will read wrong results. The ATMV2 system is used to output the voltages for tuning and matching. The LOPRO puts the probe coil into a mode where it can be tuned and matched. Some components from the ATMV2 system were omitted from the illustration to keep it more simple.

2.6.2. Automatic Tuning and Matching

The automatic Tuning and Matching is used to demonstrate the functionality of the ATMV2 system. It was performed for mechanically and electrically tuned probe coils.

Electrical Tuning and Matching

The setup used in figure 2.26 was employed for this experiment. The S_{11} value was then measured with a VNA to verify the Tuning and Matching was performed correctly. The S_{11} value was measured according to figure 2.57. A LUT was generated from 83 to 84MHz in 0.1MHz steps with the *Cryogen-3A* (*Institute of Biomedical Imaging, Graz, Austria*) and the *Cryogen-4* to verify that the proposed algorithm works for different kinds of electrical probe coils.

Mechanical Tuning and Matching

The setup used in figure 2.25 was employed for this experiment. The S_{11} value was then measured with a VNA to verify the Tuning and Matching was performed correctly. The S_{11} value was measured according to figure 2.56. First, the probe coil was manually tuned and matched to a frequency of 83MHz using the NQRduck interface. Then the LUT was generated for the frequency range from 83 to 84MHz in 0.1MHz steps. The LUT was then observed in the NQRduck program and the different values were tested using the a test function of the NQRduck program. In this test function, the steppers are driven to the positions specified in the LUT. Because the repeatability of the stepper positions is relevant, another LUT was generated in a frequency range from 83 to 87 MHz in 0.1MHz steps to verify the procedure works over a wider frequency range. The LUT was then tested in the same way as previously described.

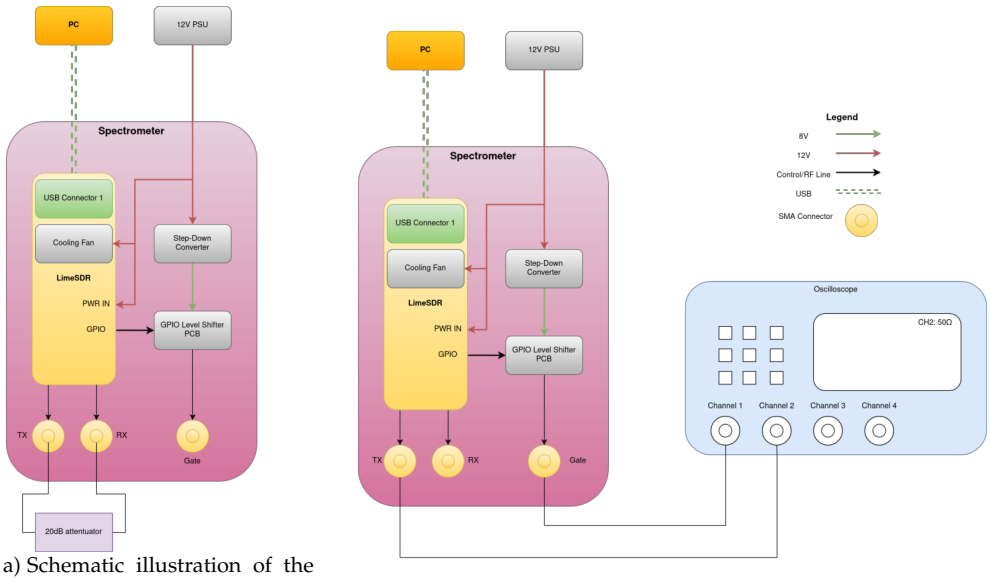
2.6.3. LimeSDR Pulse Shaping(Loopback & Oscilloscope Measurement)

The Loopback is used to test the TX and RX path of the LimeSDR. For this purpose, the TX and RX port of the LimeSDR are connected using a 20dB attenuator (Figure 2.58a). Furthermore, it is used to verify the pulse shaping capability of the spectrometer. The output pulse was further observed using an oscilloscope (UTD2102CEL, *UNI-T, Guangdong Province, China*) (Figure 2.58b).

The pulse shapes were programmed using the LimeNQR spectrometer module. The time resolution for the pulse shapes was 22 nanoseconds for all pulse shapes. The expressions from table 2.11 were evaluated from -1 to 1 for every pulse shape. A screenshot of the pulse sequence and spectrometer settings can be found in the Appendix (Figure D.1).

Name	Expression	Parameter (Symbol)
Rect	1	(None)
Sinc	$\frac{\sin(x \cdot l)}{x \cdot l}$	Scale Factor (l=3)
Gaussian	$e^{-0.5\left(\frac{(x-\mu)}{\sigma}\right)^2}$	Mean ($\mu = 0$) Standard Deviation ($\sigma = 1$)
Custom	x	(None)

Table 2.11.: List of functions with expressions and parameters.



a) Schematic illustration of the setup for the loopback.

b) Schematic illustration of the pulse shape measurement using the oscilloscope

Figure 2.58.: Illustration of the loopback and pulse shape experiment. Both experiments were conducted with various pulse shapes at 100 MHz using 100 averages. The oscilloscope triggered on the rising flank of the gate signal.

Furthermore, a more complex sequences of pulses was programmed using the NQRduck pulse programmer.

2.6.4. Sample

The sample used for verification is Triphenylbismuth ($BiPh_3$), a sample for NQR experiments. It is a well characterized sample, so it is ideal for verification. Specifically the resonance at 83.56MHz was observed because this has been thoroughly analyzed in previous implementations of the LimeSDR-based spectrometer [9].

2.6.5. Single Frequency Measurements

Single frequency measurements were performed using the LimeNQR spectrometer. Then, the same sequences were simulated using the NQRduck Simulator.

LimeNQR Measurements

The Single Frequency measurements were first performed using the LimeNQR spectrometer via the NQRduck GUI using the NQRDuck Spectrometer LimeNQR module. The mechanically tuned probe coil was Tuned and Matched to a target frequency of 83.56MHz. An S_{11} measurement was preformed using the VNA to verify the correct matching of the probe coils.

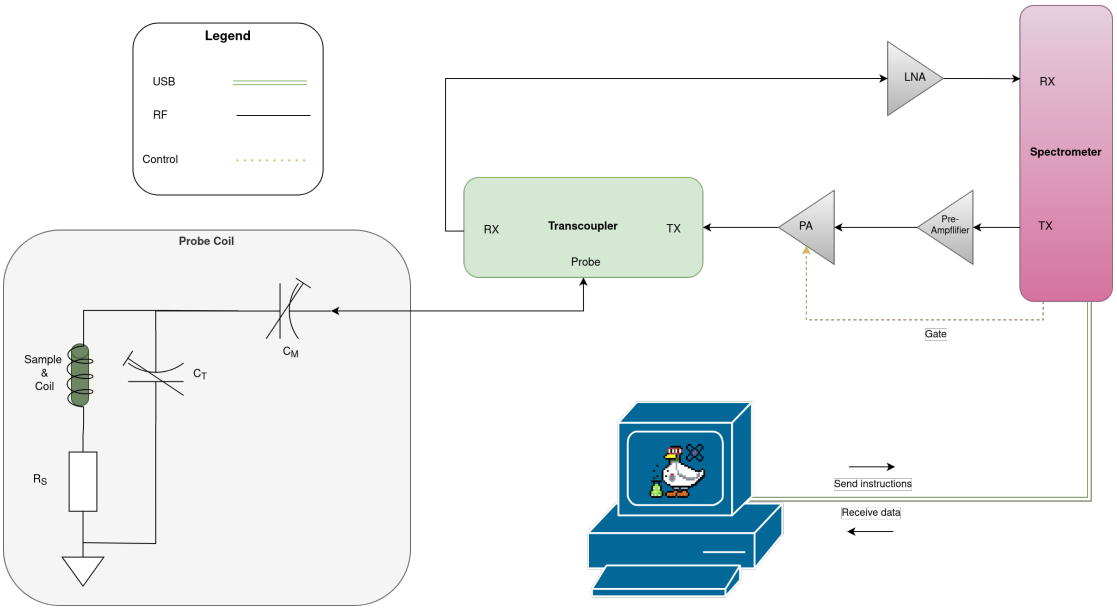


Figure 2.59.: Experimental Setup for the Single Frequency Experiments. The LimeNQR spectrometer was controlled using the NQRduck program running on a PC. The mechanically tuned probe coil was tuned and matched manually to 83.56MHz. The transcoupler was used as an RX-TX switch. A pre-amplifier was used to drive the PA from the LimeNQR spectrometer.

The single frequency measurements are used to verify the functionality of the system for magnetic resonance experiments. For this two different pulse sequences were employed.

FID A simple FID sequence was programmed using the NQRduck Pulse Programmer. To showcase the implementation of the pulse sequence in the GUI, a screenshot of the pulse sequence is depicted (Figure 2.60).

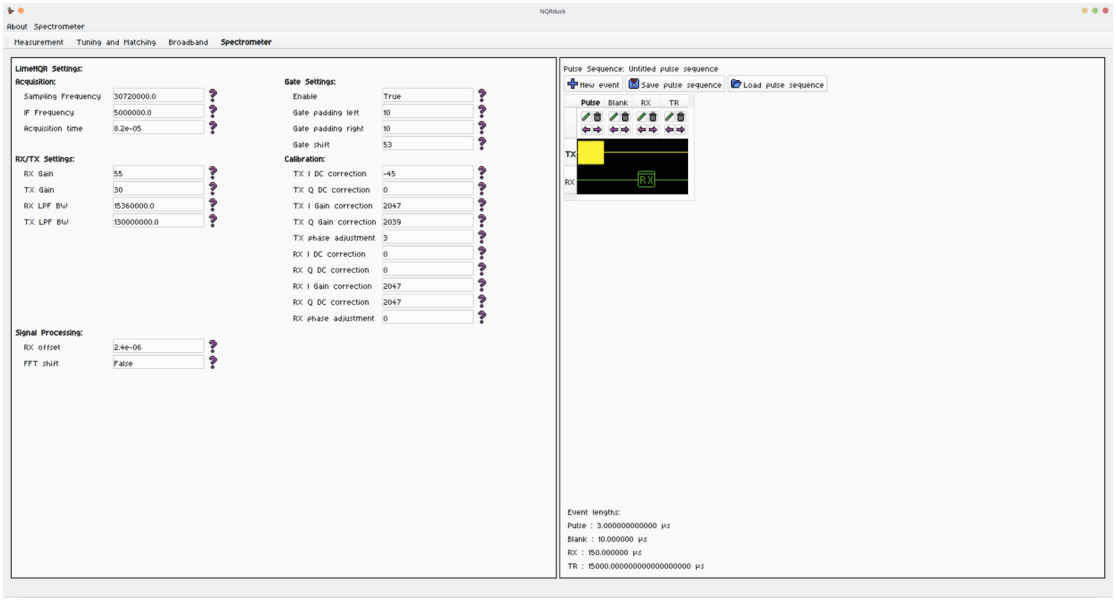


Figure 2.60.: Screenshot of the FID sequence used for single frequency measurements. The spectrometer settings are depicted on the left side and the pulse sequence is depicted on the right side. The Intermediate frequency (IF) was set to 5MHz and the acquisition time to 802 microseconds. Other settings were left at the default value for the LimeNQR spectrometer. For the pulse sequence, the excitation pulse ('Pulse') was a rectangular pulse with a length of 3 microseconds, followed by a blank time ('Blank') of 10 microseconds. The ADC readout time ('RX') was set to 150 microseconds while the repetition time ('TR') was set to 15 milliseconds.

The measurement was then performed using the NQRduck Measurement module. The number of averages was set to 1000 and the experiment frequency to 83.56MHz. The excitation pulse was optimized to yield a maximum FID signal for this sample and instrumentation.

SE The SE sequence was also programmed using the NQRduck Pulse Programmer (Figure 2.61).

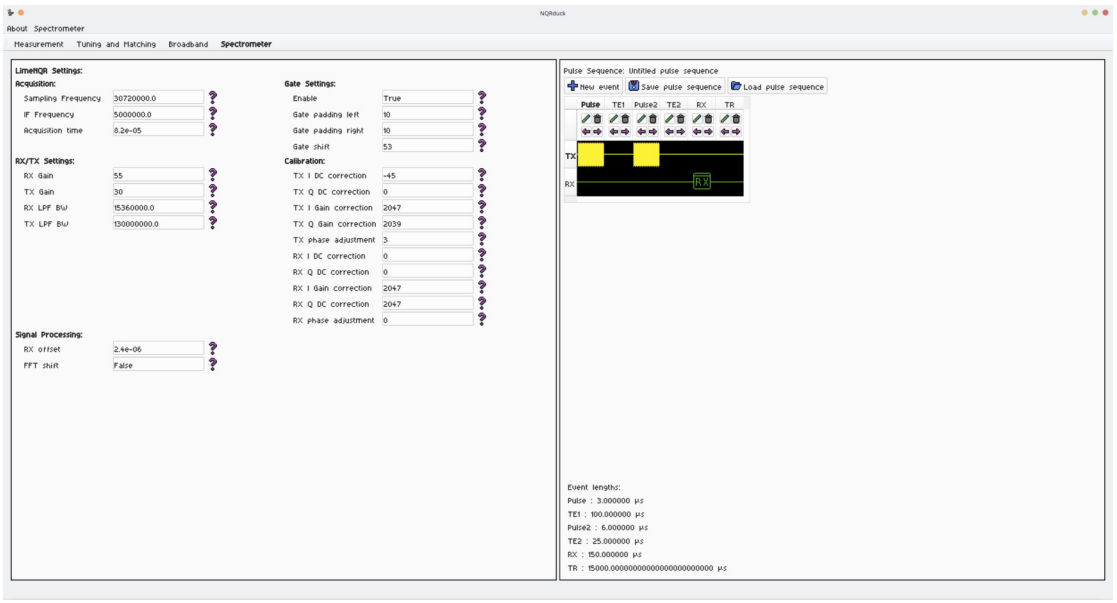


Figure 2.61.: Screenshot of the SE sequence used for single frequency measurements. The spectrometer settings are depicted on the left side and the pulse sequence is depicted on the right side. The Intermediate frequency (IF) was set to 5MHz and the acquisition time to 802 microseconds. Other settings were left at the default value for the LimeNQR spectrometer. For the pulse sequence, the first excitation pulse ('Pulse') was a rectangular pulse with a length of 3 microseconds, followed by half the echo time ('TE1') with a length of 100 microseconds. Then the inversion pulse ('Pulse2') with a length of 6 microseconds followed. The blank time ('TE2') was 25 microseconds. The ADC readout time ('RX') was set to 150 microseconds while the repetition time ('TR') was set to 15 milliseconds.

The number of averages was set to 1000 and the experiment frequency to 83.56MHz. The excitation and inversion pulse were optimized from the previous FID measurements.

Simulator

To verify the functionality of the simulator, the previously described pulse sequences for FID and SE were simulated using the NQRduck Simulator module. The settings for the simulator were taken from previously measured results [9]. Other settings like the Q factor of the probe coil were estimated using the VNA. It should be noted that the focus for the simulations should be on the integration of the simulator into the NQRduck framework. A thorough verification of the simulator should be performed on a wide variety of samples, experimental conditions and instrumentation.

The different parameters used for the simulation are depicted in Table 2.12. The number of points was chosen so that the dwell time for the single frequency measurements using the LimeNQR spectrometer and for the simulation would be the same.

Table 2.12.: Parameters and settings for the simulation.

Simulator Settings		**Experimental Setup**	
N. simulation points	4939 (FID) 8606 (SE)	Temperature (K)	300
N. of isochromats	1000	Loss TX (dB)	25
Initial magnetization	1	Loss RX (dB)	25
Gradient (mT/m)	1	Conversion factor	2884
Noise (uV)	25		
Hardware		**Sample**	
Length coil (m)	0.013	Name	BiPh3
Diameter coil (m)	0.009	Density (g/cm^3)	1.585
Number turns	6.5	Molar Mass (g/mol)	440.3
Q factor Transmit	100	Resonant freq. (Hz)	835600000.0
Q factor Receive	100	Gamma (Hz/T)	34200000.0
PA Power (W)	110	Nuclear spin	4.5
Gain	6000	Spin factor	2
		Powder factor	0.75
		Filling factor	0.2
		T1 (s)	0.000835
		T2 (s)	0.000396
		T2* (s)	5e-05

2.6.6. Broadband Measurements

For this experiment, the configurations depicted in Figure 2.26 and 2.25 were utilized, corresponding to the electrically and mechanically tuned probe coils, respectively. Lookup tables for both types of probe coils were generated using the NQRduck AutoTM module, spanning a frequency range from 83 to 84 MHz, with increments of 100 kHz for mechanically tuned probe coils and 50kHz for electrically tuned probe coils.

Spectrometer settings were configured using the NQRduck Spectrometer module, and the pulse sequence was programmed via the NQRduck Pulse Programmer module. The spectrometer settings were the same as for the single frequency measurements. Subsequently, the measurement was executed with the NQRduck Broadband module.

The FID pulse sequence, as detailed in Section 2.6.5, was employed for the mechanically tuned probe coil. For the electrically tuned probe coil, (Cryogen-4) the SE sequence described in 2.6.5 was used. The LUT was generated using the pre-defined voltages setting. The pulse length for the electrically tuned probe coil had to be increased to $16\mu s$ and $32\mu s$ because of the lower Q factor of the probe coils in the TX path. A total of 1,000 averages was set per single frequency experiment for both probe coils.

3. Results

3.1. S_{11} measurement

For the S_{11} measurements, the probe coils were manually tuned and matched. The S_{11} magnitude and phase was then measured using a VNA and the NQR-duck program in combination with the ATMV2 system.

3.1.1. Mechanically Tuned Probe Coil

The probe coil was tuned and matched to 83.56MHz using the VNA. The experimental setup was employed according to Figure 2.56.

The mechanically tuned probe coil was measured in the frequency range from 78.56 to 88.56 MHz with 401 sweep points using the VNA (Figure 3.1) and NQRduck with the ATMV2.

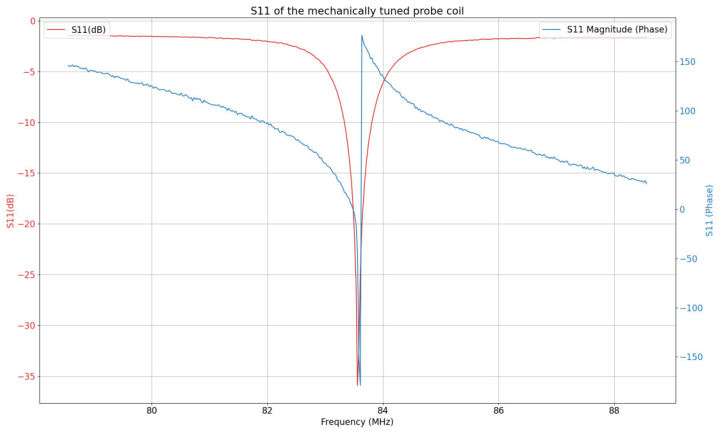


Figure 3.1.: S_{11} measurement of the mechanically tuned probe coil measured with the VNA from 78.56 to 88.56 MHz and plotted using Matplotlib.

For the NQRduck, the measurement was conducted with the attenuator connected (Figure 3.2).

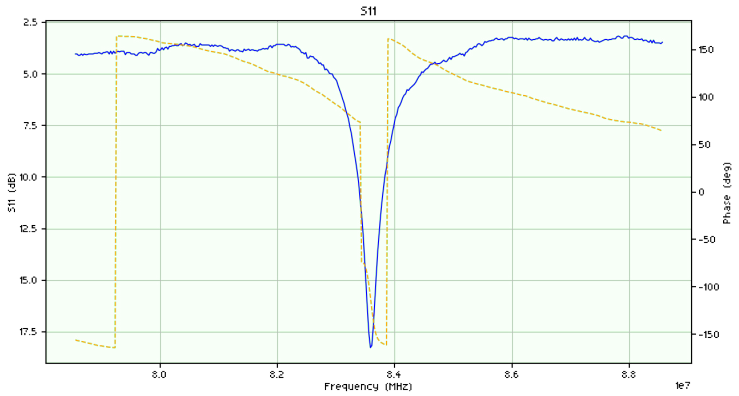


Figure 3.2.: S_{11} measurement of the mechanically tuned probe coil using the NQRduck from 78.56 to 88.56 MHz. With the -10dB attentuator connected.

The same measurement was then conducted without the attenuator (Figure 3.3).

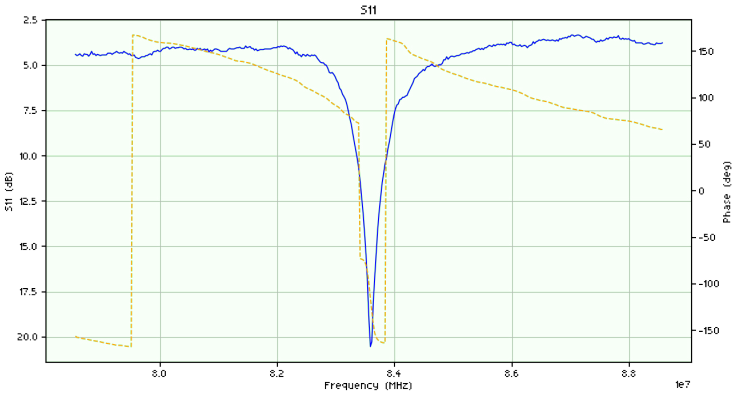


Figure 3.3.: S_{11} measurement of the mechanically tuned probe coil using the NQRduck. With the -10dB attentuator not connected.

The S_{11} measurement was then repeated in a frequency range from 82.56 to 84.56MHz with 401 sweep points using the NQRduck with the ATMV2. This measurement was conducted to provide a finer frequency resolution (Figure 3.4.

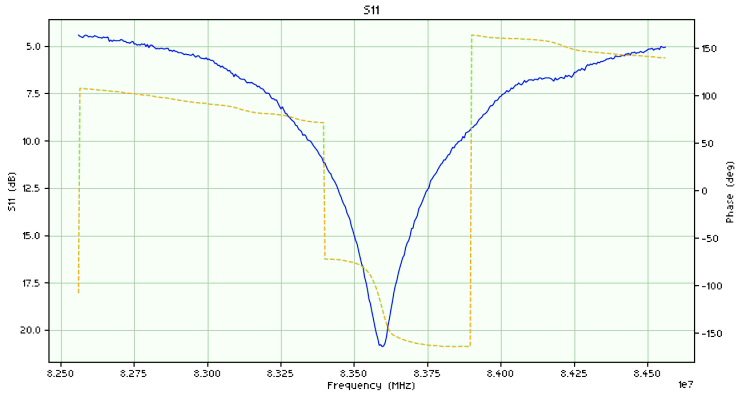


Figure 3.4.: S_{11} measurement of the mechanically tuned probe coil using the NQRduck. With the -10dB attentuator not connected.

3.1.2. Electrically Tuned Probe Coils

The electrically tuned probe coil was tuned and matched to 83.56MHz using the VNA. The voltages were set using the ATMV2 system. The tuning voltage was 0.85V and the matching voltage 0.45V.

The electrically tuned probe coil was measured in the frequency range from 78.56 to 88.56MHz with 401 sweep points using the VNA (Figure 3.5).

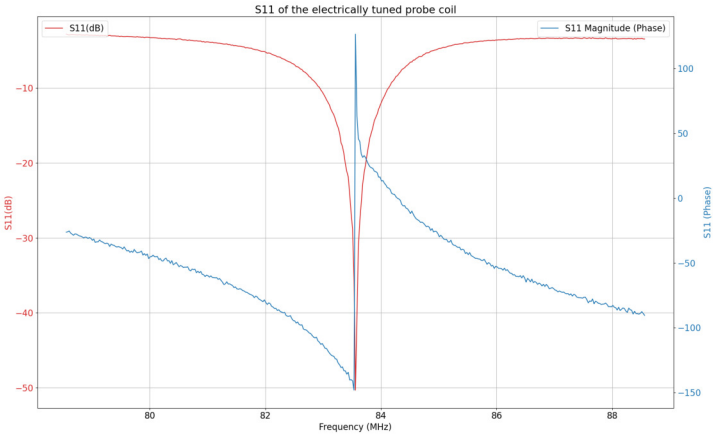


Figure 3.5.: S_{11} measurement of the electrically tuned probe coil measured with the VNA from 78.56 to 88.56 MHz and plotted using Matplotlib.

The S_{11} measurement was then conducted using the NQRduck in combination with the ATMV2 system (Figure 3.6) using the same frequency range.

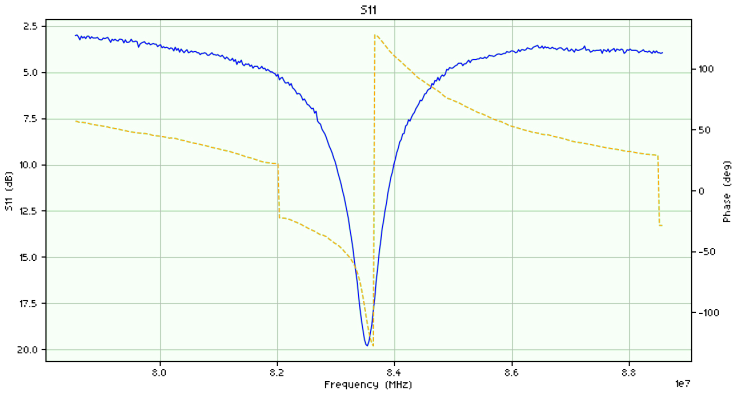


Figure 3.6.: S_{11} measurement of the electrically tuned probe coil using the NQRduck from 78.56 to 88.56 MHz

For this experiment, no measurement without attenuator was conducted because this would detune the probe coils.

The electrically tuned probe coil was measured in the frequency range from 82.56 to 84.56 MHz with 401 sweep points using the VNA (Figure **??**) and NQRduck with the ATMV2.

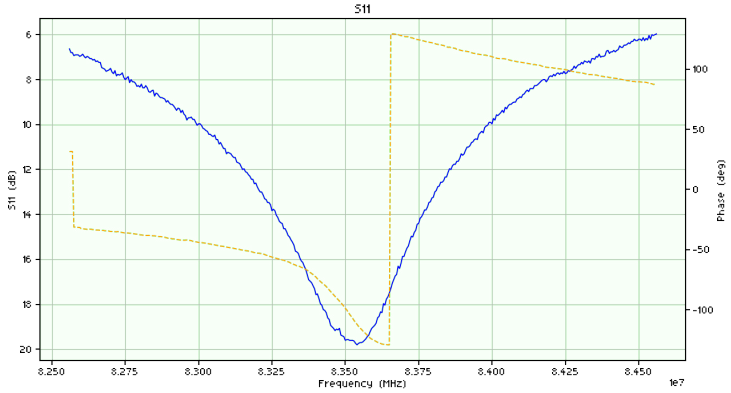


Figure 3.7.: S_{11} measurement of the electrically tuned probe coil using the NQRduck from 82.56 to 84.56 MHz

3.2. Automatic Tuning and Matching

The tuning and matching experiments were conducted to showcase the tuning and matching capability of the ATMV2 system for electrically- and mechanically tuned probe coils.

3.2.1. Electrical Tuning and Matching

Cryogen-3A A LUT was generated from 83MHz to 84MHz with a step size of 0.1MHz. The resulting reflection coefficient was then measured using the VNA. The measured values were plotted using Matplotlib. The measured S_{11} values are shown in Figure 3.8.

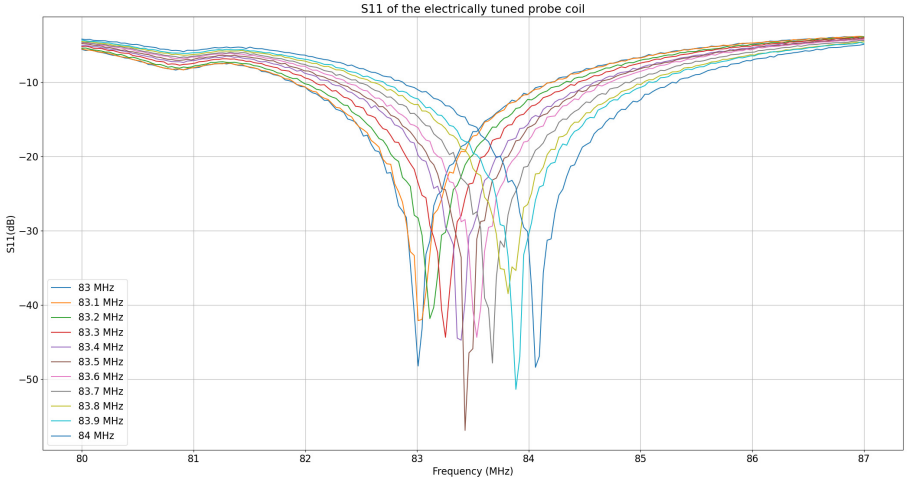


Figure 3.8.: S_{11} measurement using the VNA of the electrically tuned probe coil after automatic tuning and matching in a frequency range from 83 MHz to 84MHz in 0.1MHz steps. The data was then plotted using Matplotlib.

Plot 3.9 shows the S_{11} values at the frequency where the LUT was generated. Table 3.1 depicts the Tuning and Matching voltages for each frequency.

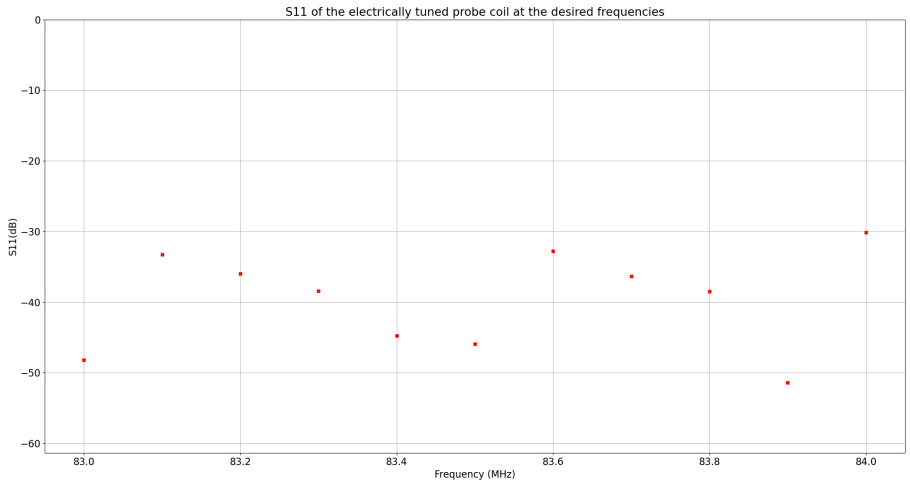


Figure 3.9.: S_{11} values of the electrically tuned probe coil after automatic Tuning and Matching at the frequencies where the LUT was generated. The S_{11} values were measured using a VNA and plotted using Matplotlib.

Table 3.1.: Lookup table for each frequency with the Tuning and Matching voltages.

Frequency (MHz)	Tuning Voltage (V)	Matching Voltage (V)
83.0	1.41	1.58
83.1	1.42	1.58
83.2	1.43	1.55
83.3	1.43	1.59
83.4	1.44	1.59
83.5	1.44	1.62
83.6	1.44	1.66
83.7	1.46	1.61
83.8	1.47	1.61
83.9	1.47	1.64
84.0	1.48	1.66

Cryogen-4 The *Cryogen-4* probe coil was tuned and matched in a frequency range from 83 to 84 MHz using the pre-defined voltage algorithm with $0.66V$ for the tuning voltage and $0.04V$ for the matching voltage. The resulting LUT was the measured using a VNA (Figure 3.10 & 3.11).

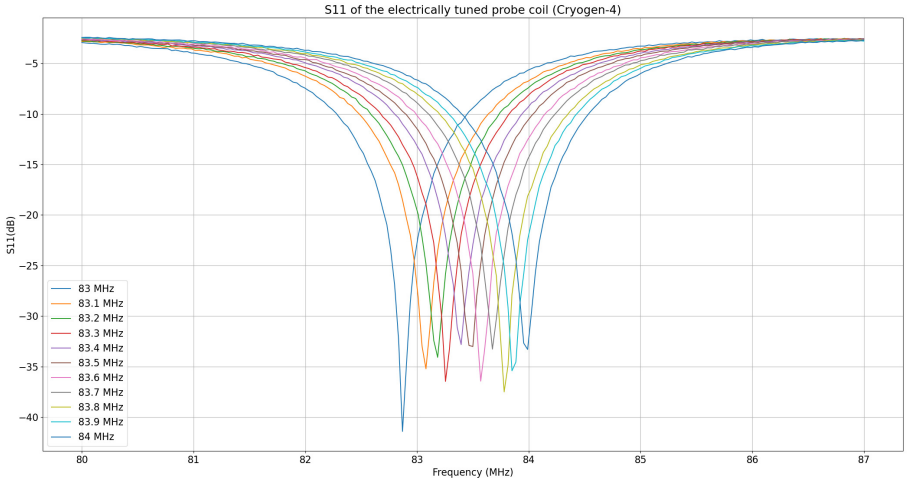


Figure 3.10.: S_{11} measurement using the VNA of the electrically tuned probe coil after automatic tuning and matching to in a frequency range from 83 MHz to 84MHz in 0.1MHz steps. The data was then plotted using Matplotlib.

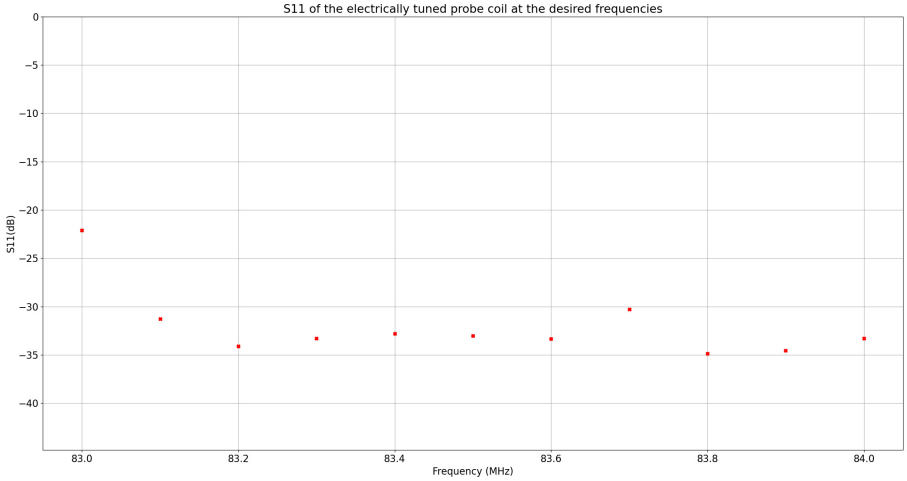


Figure 3.11.: S_{11} values of the electrically tuned probe coil after automatic Tuning and Matching at the frequencies where the LUT was generated. The S_{11} values was measured using a VNA and plotted using Matplotlib.

The voltages of the LUT are depicted in Table 3.2.

Table 3.2.: Lookup table for each frequency with the Tuning and Matching voltages.

Frequency (MHz)	Tuning Voltage (V)	Matching Voltage (V)
83.0	0.61	0.09
83.1	0.64	0.13
83.2	0.68	0.18
83.3	0.71	0.22
83.4	0.73	0.33
83.5	0.78	0.33
83.6	0.82	0.28
83.7	0.82	0.44
83.8	0.85	0.51
83.9	0.89	0.49
84.0	0.91	0.55

3.2.2. Mechanical Tuning and Matching

Automatic Tuning and Matching: 83 to 84MHz A LUT was generated from 83MHz to 84MHz with a step size of 0.1MHz. This resulting reflection coefficient was then measured using the VNA. The measured values were plotted using Matplotlib. The measured S_{11} values are shown in Figure 3.12.

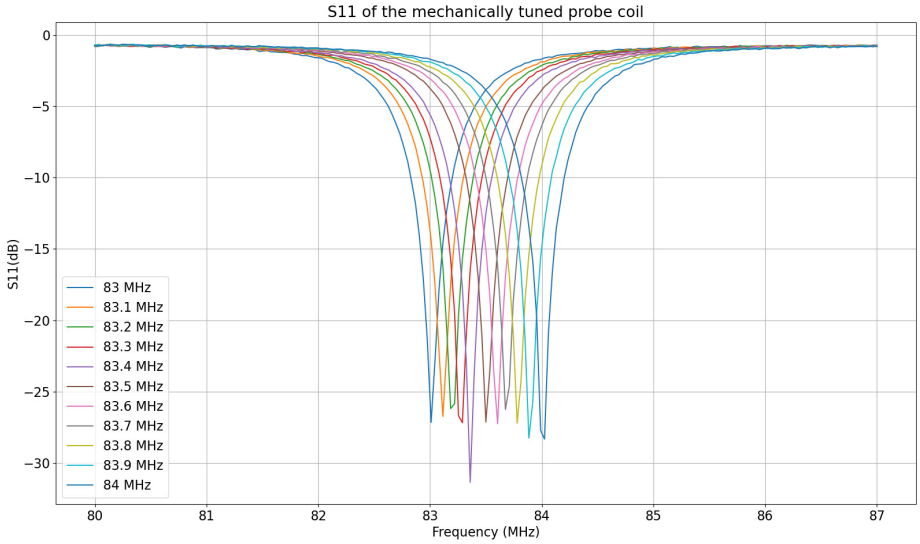


Figure 3.12.: S_{11} measurement using the VNA of the mechanically tuned probe coil after automatic tuning and matching in a frequency range from 83 MHz to 84MHz in 0.1MHz steps. The data was then plotted using Matplotlib.

Plot 3.13 shows the S_{11} values at the frequency where the LUT was generated and the broadband measurement would be performed at.

Table 3.3 depicts the Tuning and Matching stepper positions for each frequency.

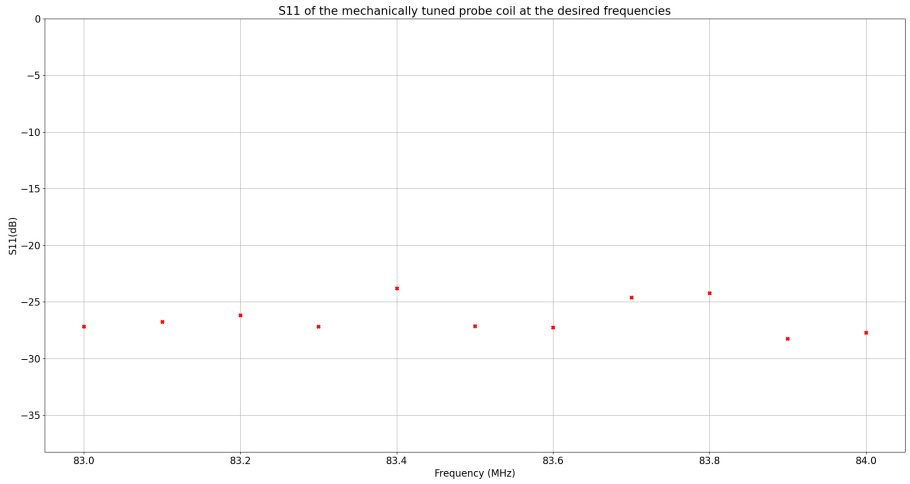


Figure 3.13.: S_{11} values of the mechanically tuned probe coil after automatic Tuning and Matching at the frequencies where the LUT was generated. The S_{11} values were measured using a VNA and plotted using Matplotlib.

Table 3.3.: Lookup table for each frequency with the Tuning and Matching positions of the stepper motors.

Frequency (MHz)	Tuning Position	Matching Position
83.00	31370	88250
83.10	31390	87900
83.20	31410	87400
83.30	31430	87050
83.40	31450	86700
83.50	31490	87200
83.60	31510	86700
83.70	31530	86800
83.80	31550	86300
83.90	31570	85900
84.00	31590	85400

Automatic Tuning and Matching: 83-87MHz A LUT was generated from 83MHz to 87MHz with a step size of 0.1MHz. This resulting reflection coefficient was then measured using the VNA. The measured values were plotted using Python. The measured S_{11} values are shown in Figure 3.12.

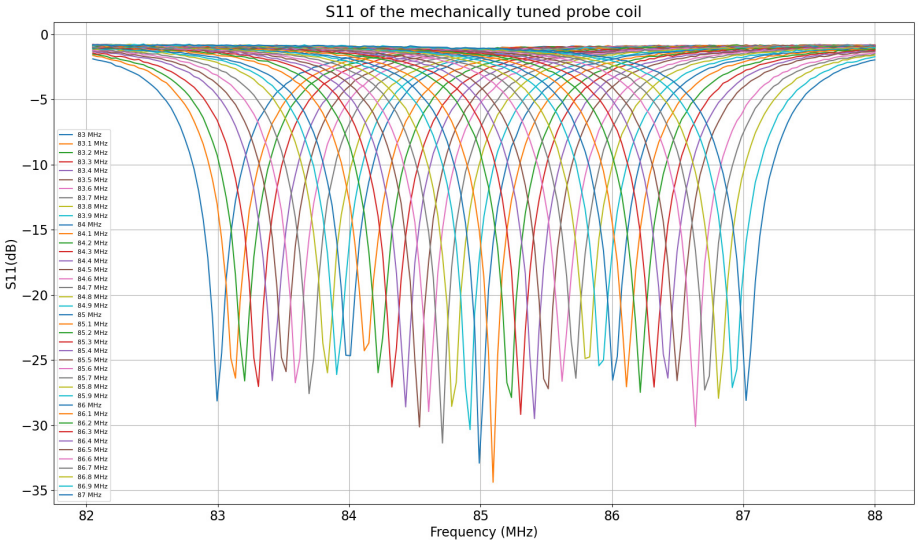


Figure 3.14.: S_{11} measurement using the VNA of the mechanically tuned probe coil after automatic Tuning and Matching in a frequency range from 83 MHz to 87MHz in 0.1MHz steps. The data was then plotted using Matplotlib.

Plot 3.15 shows the S_{11} values at the frequency where the LUT was generated and the broadband measurement would be performed at.

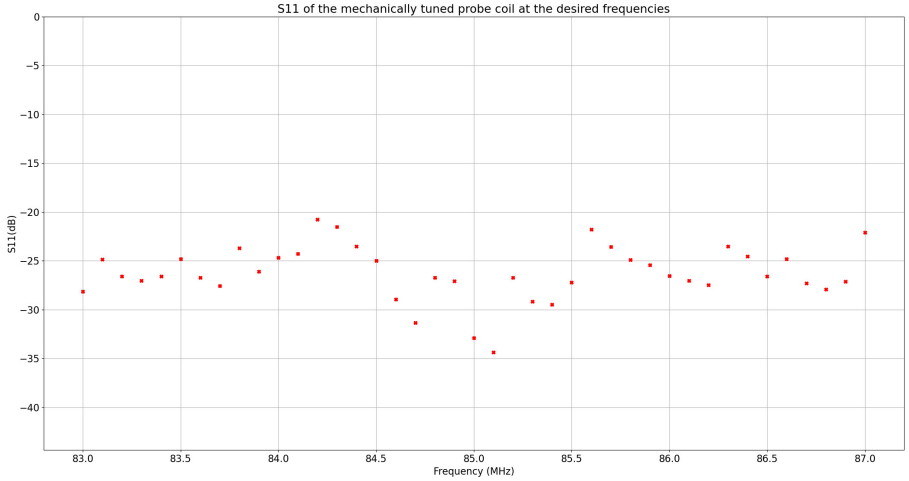


Figure 3.15.: S_{11} values of the mechanically tuned probe coil after automatic Tuning and Matching at the frequencies where the LUT was generated. The S_{11} values were measured using a VNA and plotted using Matplotlib.

3.3. LimeNQR: Pulse Shaping

The pulse shaping experiments were conducted to showcase the pulse shaping capability of the NQRduck spectrometer module.

3.3.1. Loopback

The loopback setup was employed according to Figure 2.58a. The pulse shapes were programmed using the NQRduck spectrometer module. A description of the pulses can be found in table 2.11. The spectrometer settings can be found in the Appendix D.1. The pulse frequency was 100MHz and 100 averages were recorded.

Time and frequency domain plots of the rectangular loopback pulse recorded with the LimeNQR spectrometer (Figure 3.16 & 3.17).

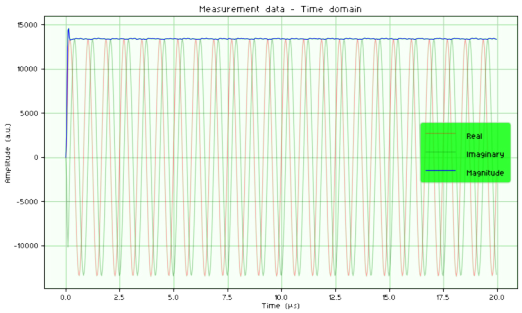


Figure 3.16.: Time Domain Rectangular Pulse

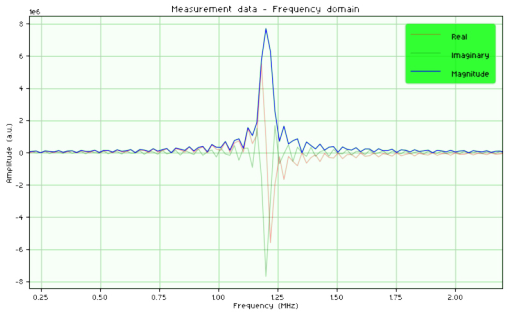


Figure 3.17.: Frequency Domain Rectangular Pulse

Time and frequency domain plots of the sinc loopback pulse recorded with the LimeNQR spectrometer (Figure 3.18 & 3.19).

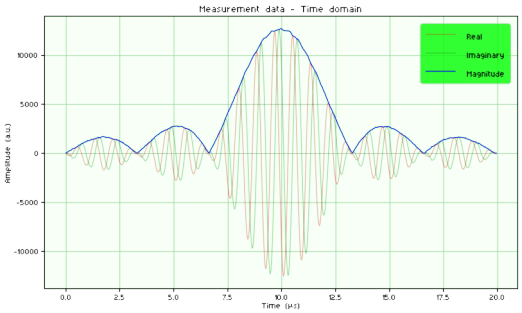


Figure 3.18.: Time Domain Sinc Pulse

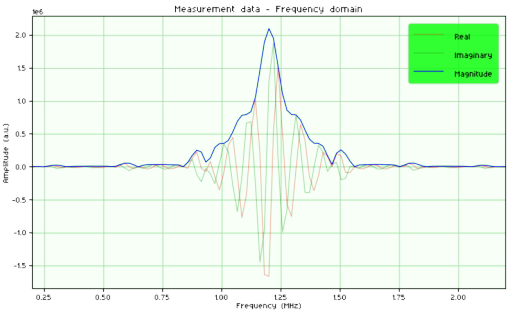


Figure 3.19.: Frequency Domain Sinc Pulse

Time and frequency domain plots of the Gaussian loopback pulse recorded with the LimeNQR spectrometer (Figure 3.20 & 3.21).

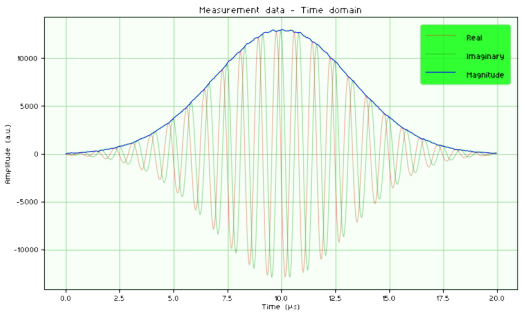


Figure 3.20.: Time Domain Gauss Pulse

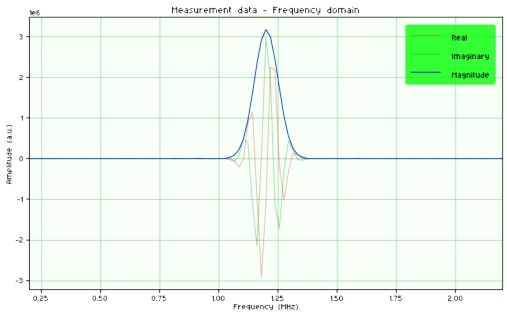


Figure 3.21.: Frequency Domain Gauss Pulse

Time and frequency domain plots of the custom loopback pulse recorded with the LimeNQR spectrometer (Figure 3.22 & 3.23).

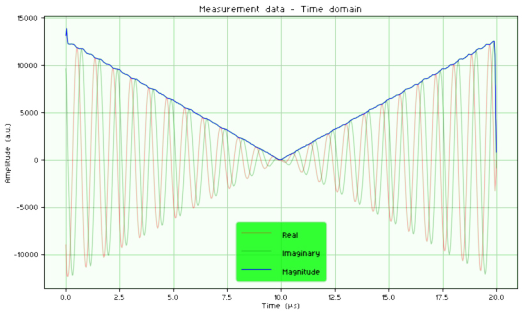


Figure 3.22.: Time Domain Custom Pulse

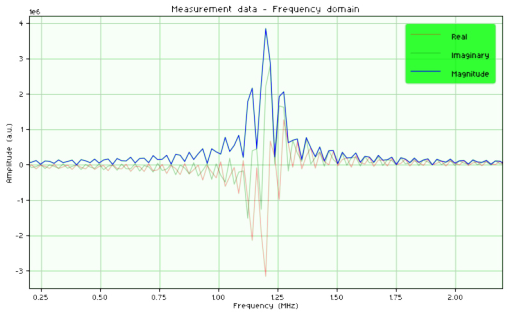
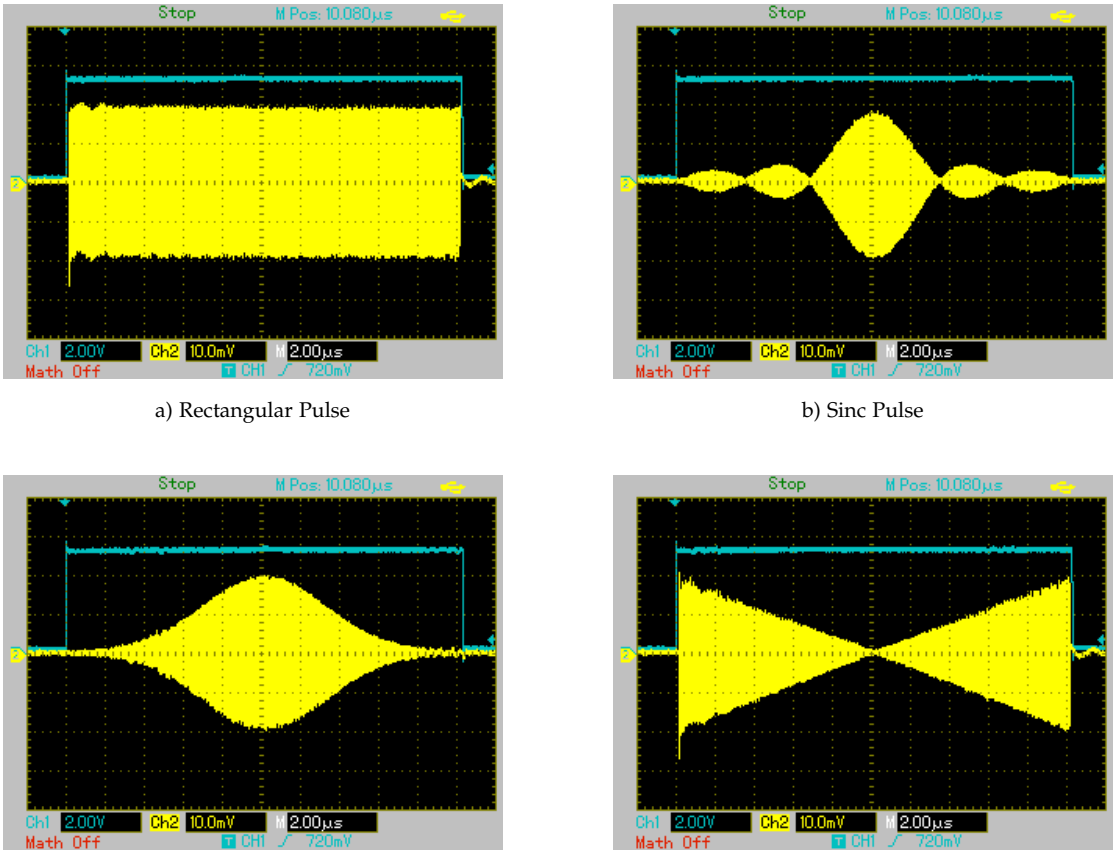


Figure 3.23.: Frequency Domain Custom Pulse

3.3.2. Oscilloscope Measurement

The oscilloscope measurements were conducted according to Figure 2.58b. The different figures show the different pulse shapes (Figure 3.24) recorded with the oscilloscope. The pulses had a frequency of 100MHz.



a) Rectangular Pulse

b) Sinc Pulse

c) Gaussian Pulse

d) Custom Pulse

Figure 3.24.: Oscilloscope measurements of the different pulse shapes. Channel 1 (yellow) is the pulse signal. Channel 2 (blue) is the Gate signal and was used as trigger for the measurement.

A more complex pulse train was created using the NQRduck Pulse Programmer (Figure 3.25). To showcase the intuitive visualization of the pulse sequence, a screenshot of the programmed pulse sequence is shown in comparison to the pulse train measured with an oscilloscope (Figure 3.26).

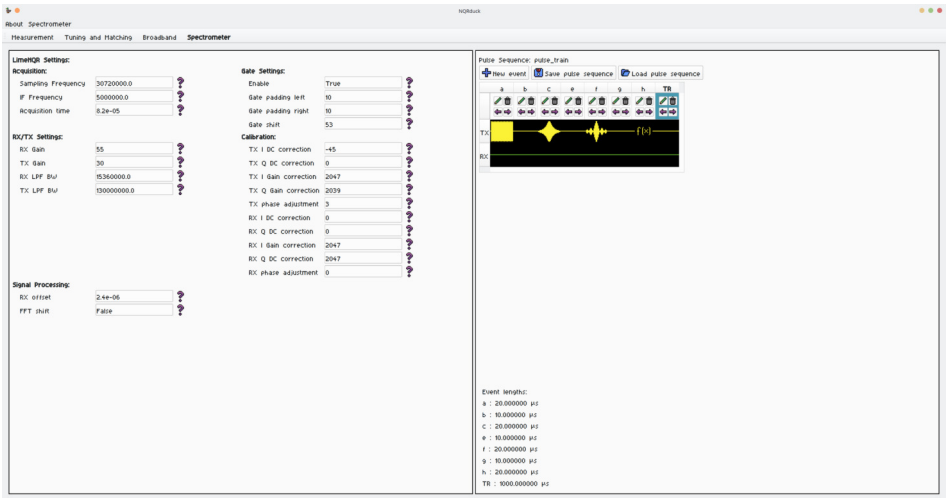


Figure 3.25.: Screenshot of the spectrometer module where the pulse train was programmed. The pulse sequence is depicted on the right side. There are four different pulse shapes (Rectangular, Sinc, Gaussian and Custom) with a length of 20 microseconds each. The pulses are separated by a blank time of 10 microseconds.

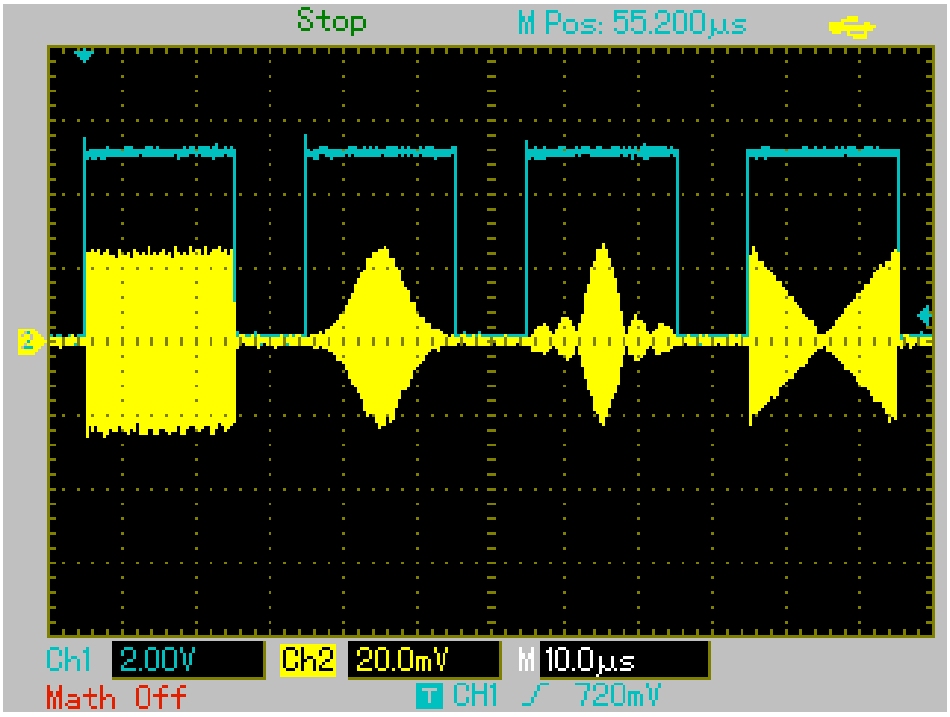


Figure 3.26.: Pulse train recorded with an oscilloscope. The blue signal (Channel 1) is the gate signal of the LimeNQR. The yellow signal (Channel 2) is the pulse shape output via the TX path of the LimeNQR.

3.4. Single Frequency Measurements

3.4.1. LimeNQR

FID

The result for the Single Frequency Measurement of the $BiPh_3$ sample is depicted in Figure 3.27 (Spectrum) and Figure 3.28 (Time Domain).

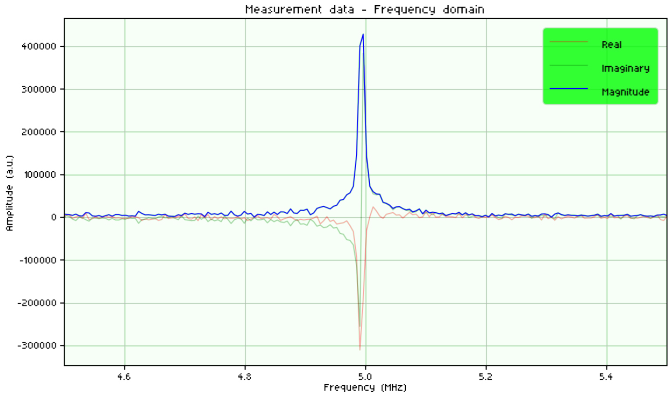


Figure 3.27.: FID Spectrum of the $BiPh_3$ sample zoomed in to ± 1 MHz around the sample peak. The pulse was recorded at an Intermediate frequency (IF) of 5MHz. No phase correction was applied to the data points.

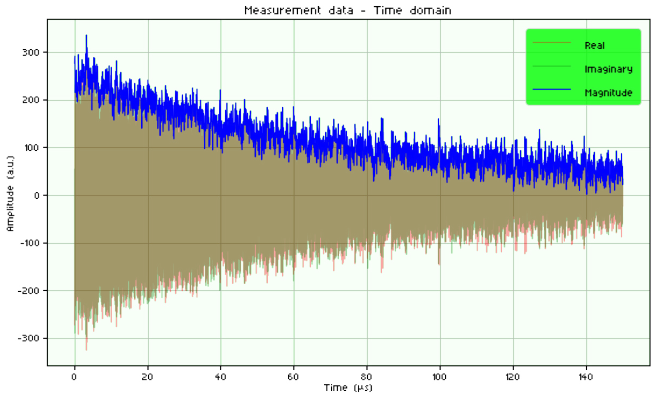


Figure 3.28.: Time Domain View of the FID for the $BiPh_3$ sample.

The full spectrum can be found in the Appendix D.2.

SE

The result of the SE sequence is depicted in figure 3.30 (Spectrum) and Figure 3.30 (Time Domain).

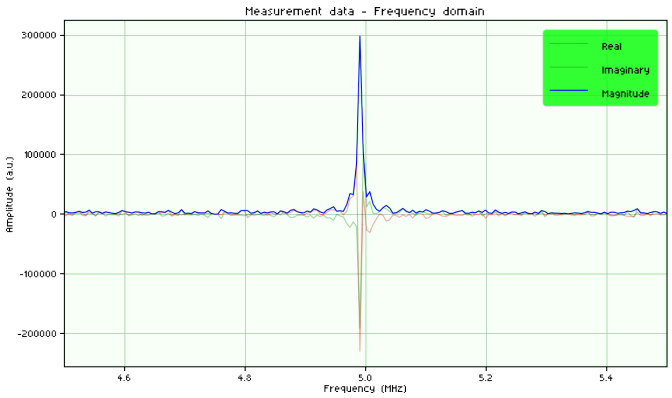


Figure 3.29.: SE Spectrum of the $BiPh_3$ sample zoomed in to ± 1 MHz around the sample peak. The pulse was recorded at an Intermediate frequency (IF) of 5MHz. No phase correction was applied to the data points.

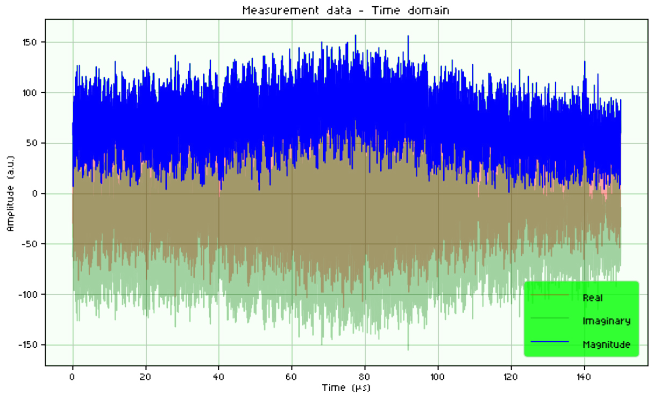


Figure 3.30.: Time Domain View of the SE for the $BiPh_3$ sample.

The full spectrum can be found in the Appendix D.3.

3.4.2. Simulator

3.4.3. FID

The result of the FID simulation for the $BiPh_3$ sample is depicted in Figure 3.31 (Spectrum) and Figure 3.32 (Time Domain).

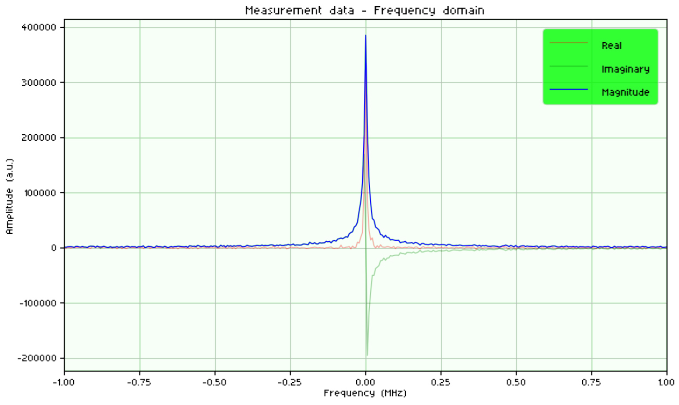


Figure 3.31.: Simulated FID Spectrum of the $BiPh_3$ sample.

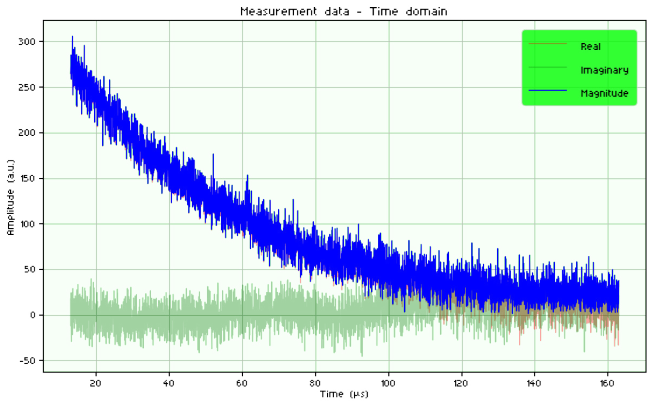


Figure 3.32.: Time Domain View of the simulated FID for the $BiPh_3$ sample.

The full spectrum can be found in the Appendix D.4.

SE

The result of the SE simulation for the $BiPh_3$ sample is depicted in Figure 3.33 (Spectrum) and Figure 3.34 (Time Domain).

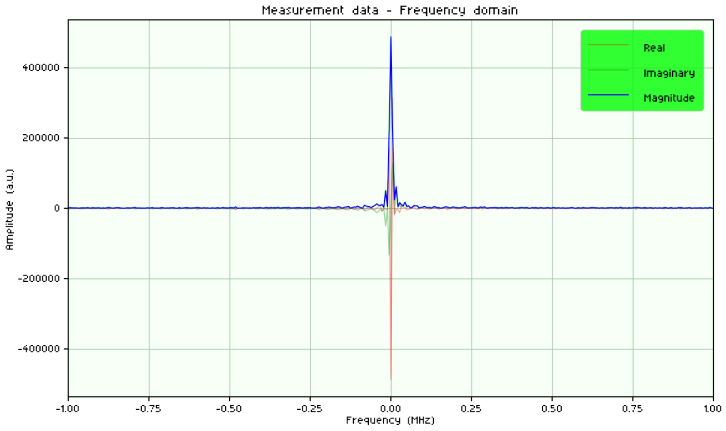


Figure 3.33.: Simulated SE Spectrum of the $BiPh_3$ sample.

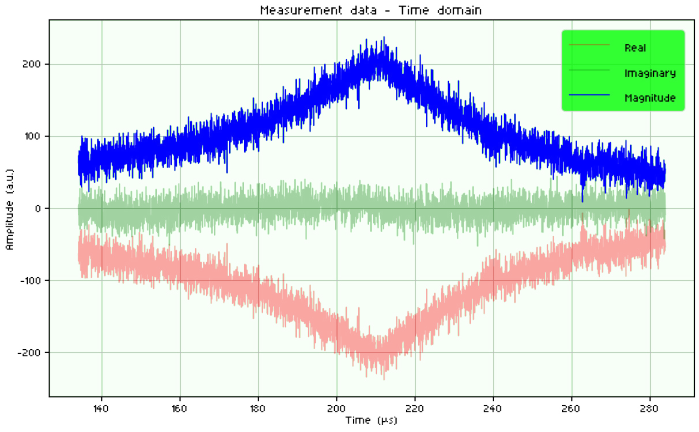


Figure 3.34.: Time Domain View of the simulated SE for the $BiPh_3$ sample.

The full spectrum can be found in the Appendix D.5.

3.5. Broadband Measurements

3.5.1. Electrically Tuned Probe Coil

A $BiPh_3$ sample (5mm tube) was measured in a frequency range from 83 to 84MHz with a step size of 50kHz using a SE sequence (Figure 3.35).

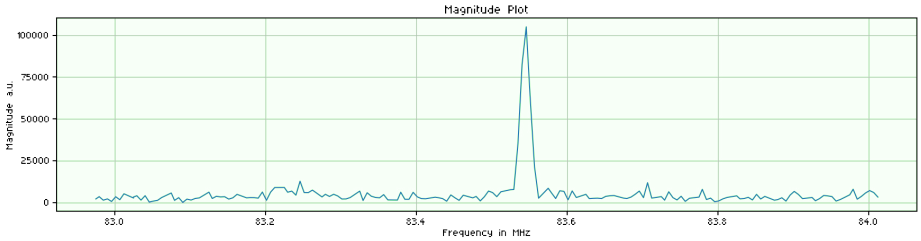


Figure 3.35.: Broadband Scan with the Electrically Tuned Probe Coil ($BiPh_3$). A peak from the sample can be seen at 83.54MHz.

3.5.2. Mechanically Tuned Probe Coil

The $BiPh_3$ sample was measured in a frequency range from 83 to 84MHz with a step size of 100kHz using a FID sequence (Figure 3.36).

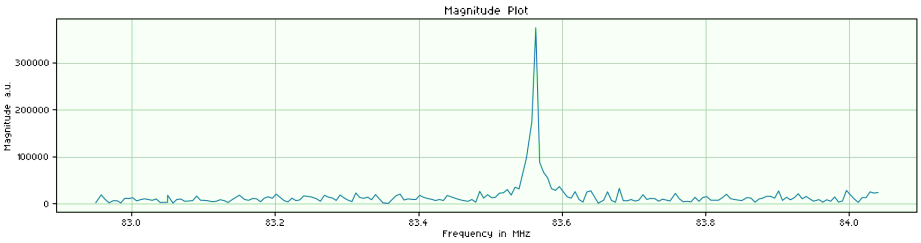


Figure 3.36.: Broadband Scan with the Mechanically Tuned Probe Coil ($BiPh_3$). A peak from the sample can be seen at 83.56MHz.

4. Discussion

4.1. NQRduck Software

Software Architecture The final implementation of NQRduck software proves to be highly customizable, with data and functionality encapsulated in discrete modules, thereby rendering the programming of individual modules manageable.

Because of the object-oriented design approach, implementation of new modules, especially spectrometer submodules, showed to be highly efficient. The ability to automatically generate views and use defined software interfaces avoided duplication of code. The flexibility in designing the GUI — both through design tools such as Qt Designer and direct code — allows rapid and efficient development of new modules. Automatic loading of NQRduck modules via entry points eliminates the need for continually updating configuration files. This approach substantially simplifies the module loading process and mitigates the risk of unforeseen bugs.

The capability to selectively (un)install modules, thereby tailoring the functionality of the NQRduck program, is advantageous for educational settings. In contexts such as student laboratories, installing only the requisite modules prevents confusion by reducing the GUI to the essential functionality.

Software Testing As with any GUI program, the emergence of bugs is an inevitable challenge. Rigorous tracking of bugs and structured software testing should be employed in the future maintanance of the project. The NQRduck program was primarily tested under Linux-based systems (Manjaro, Kubuntu). For these systems, it was observed that switching between different windowing systems lead to minor bugs in the GUI. The windowing systems X11 [1] and Wayland [2] were tested. When using X11, the GUI showed improved performance and fewer bugs compared to Wayland. The GUI was not thoroughly tested with Windows-based systems.

Future Work Future work may entail entrusting students with the implementation of their own module within the NQRduck software. This could be a good test to see how well this framework is received by students. Furthermore it

[1] https://www.x.org/wiki/

[2] https://wayland.freedesktop.org/

should be tested in student labs where students could use it under supervision. Such an exercise could also reveal areas where GUI design could be refined and potential bugs addressed.

Another interesting aspect would be the implementation of automated pulse sequences to implement advanced techniques like Inversion Recover sequences. This could be easily implemented by evaluating variables in the *PulseSequence* object. An implementation of automation of pulse sequences has been started within the NQRduck Spectrometer module. One could also implement the option to specify pulse sequences in code similar to Pulseq [3]. This would allow for highly customizable pulse sequences.

4.2. S_{11} Measurement

The S_{11} measurement is fundamental to the system's tuning and matching functionalities. The various improvements through soft- and hardware upgrades significantly enhanced the performance of the system. However, the ATMV2 still relies on relatively low-cost components - totaling less than 300 Euros, with the directional couplers being the priciest elements. Consequently, the performance is not as good as more advanced instruments like the Rhode & Schwarz ZVL 3 VNA.

The speed of the S_{11} frequency sweeps was improved by various software tweaks. Right now, a frequency sweep with 201 points takes about 4 seconds.

Limitations The system's limitations in S_{11} measurement are evident when examining the mechanical probe coil measurements with and without an attenuator (Figure 3.5 and 3.3).

The limitations arise with probe coils that are well-matched, where the reflected signal levels tend to be low. The use of an attenuator sets the output signal at roughly -15dBm, detaching the forward wave at an estimated -35dBm. Taking into account an insertion loss of about -5dB for the entire system, reflected signals from coils with an S_{11} value better than -20dB fall below the AD8302's input range (Table 2.5). By removing the attenuator, an improvement can be seen in the minimum detectable reflection strength to around -25dB (Figure 3.3).

Unfortunately, this method isn't feasible for electrically tuned probe coils, as signal levels above -15dB detune the RX pathway. A potential solution could be the addition or removal of an attenuator in the measurement chain via a supplementary set of RF switches, depending on the specific probe coil under assessment. Alternatively, opting for a directional coupler with a different coupling factor, such as 10dB, could be a viable approach.

[3] https://pulseq.github.io/

Further limitations arise from the number of points for one scan. This is evident when observing figures 3.4 & 3.7. Since the number of points is related to the frequency resolution, using high-Q probe coils could result in a resolution that is not fine enough to detect the minimum S_{11} value, particularly if the frequency range is too broad. The number of points per scan can be increased to improve resolution, but this results in a linear increase in scan duration.

Phase Correction The phase correction algorithm is fundamentally effective, yet it is notably sensitive to noise and signal boundary conditions. The algorithm could be improved by extensively testing it under various conditions and tweaking parameter values. Another solution would be to perform two measurements with a 90 degree phase shift between those measurements. Through the difference of these measurements, one could then determine the sign of the phase. This could be a more reliable method to determine the phase sign compared to software algorithms.

Additionally, when comparing the phase measurement of the VNA (Figure 3.1) and of the ATMV2 (Figure 3.2) an offset error can be observed. This error likely stems from the lack of calibration.

Calibration In the current implementation, the ambiguity of the phase correction makes the calibration unreliable. If this issue could be fixed, it could significantly improve the performance of the S_{11} measurements. Furthermore, the assumption that the calibration adapters are ideal and do not exhibit parasitic inductance and capacitance is questionable. Commercial VNAs usually assume some kind of equivalent circuit with known parasitics for the different adapters. While this approximation may hold for lower frequencies (below 300MHz), validating this assumption through empirical testing is advisable to ensure measurement accuracy across the full operational spectrum.

4.3. Automatic Tuning and Matching

The Tuning and Matching capability was improved in comparison to the previous implementation. Especially the addition of the RF switch, which allows for easy changing between the magnetic resonance experiment setup and tuning and matching setup, is very convenient. Furthermore, the Tuning and Matching is more reliable compared to the previous implementation because of the improved S_{11} measurements. The control of the ATMV2 with a GUI, compared to only over a serial monitor, is also very convenient. The main limitation of the system is the maximum achievable S_{11} value of -25dB.

Mechanically Tuned Probe Coils For the mechanically tuned probe coils, it is still unclear if the sensorless end point detection for the tuning and matching

capacitor, by driving them to their maximum values, damages them in any way. Over the course of the last two years, since developing the system in the course of the Bachelor's project, the matching capacitor had to be replaced once. However, it is unclear if this damage had been present before the initial implementation, because problems were already observed back then [24].

The system works for Tuning and Matching of mechanically tuned probe coils (Figure 3.13). However, the mechanical setup is not ideal. The tuning stepper especially suffers from strong gear backlash. When changing direction, an approximate 60 steps have to be taken in order for the stepper to start moving the tuning capacitor again. For reference, the number of steps used to tune the probe coil by about 100kHz is approximately 20 steps (Figure 3.3).

While the backlash was corrected inside the NQRduck software by tracking the last direction of rotation of the stepper motor, this limits the accuracy of the system and can degrade the Tuning and Matching of the probe coil over a larger frequency range (Figure 3.14).

In a first step, a more accurate mechanical setup should be implemented. In further work, it could then be observed if the backlash in clockwise and counterclockwise direction differ and the algorithms could be adjusted accordingly.

Electrically Tuned Probe Coils The speed and accuracy of the Tuning and Matching was improved compared to previous implementations [9]. Especially the algorithm described in Figure 2.33 improved the speed and allowed the system to scan with a finer final voltage resolution. The integration of all components in one portable system (DAC & reflectometer) is very convenient.

The Tuning and Matching process reliably achieved S_{11} values for both probe coils below -30dB (Figures 3.8 & 3.10).

The Tuning and Matching of electrically tuned probe coils works (Figure 3.8). However, this approach assumes the electrically tuned probe coil maintains its RF properties during the Tuning and Matching process, with no changes due to issues such as faulty connections. The electrical probe coils often show faulty connectors because of the mechanical stress applied to the SMA and RJ45 connectors over the course of several years.

Furthermore, the sample coil was built so that it could be changed out very quickly. However, this was not ideal for the experiments conducted in the course of this project. The sample coil often showed faulty connection during the measurement, making the broadband measurements with electrically tuned probe coils highly unreliable. Error detection would therefore be especially important for electrically tuned probe coils as these have shown numerous problems during testing.

In future additions, the ATMV2 system could also replace the LOPRO for the control voltages. This could make the system even more portable and convenient to use.

4.4. Pulse Shaping

The pulse shaping capability of the NQRduck software allows for easy programming of pulse shapes. This is a great feature in the context of the didactic application. In the loopback plots, the different spectral shapes of the different pulse shapes can be seen in a very intuitive way (Figures 3.16 & 3.17 - 3.23).

Initially the software provided with the limr package had a bug which limited the maximum number of arguments that could be sent to the C++ program. This in turn led to a limitation in the length of the pulse sequence. This bug was fixed in the course of this project so pulse sequences are no longer limited in length.

4.5. Single Frequency Measurements

The NQRduck software allows for easy programming and execution of single frequency experiments. The programming of the pulse sequences is very easy and straightforward. The GUI is designed in a way that the final pulse sequence resembles the pulse sequences as they are depicted in textbooks, which is beneficial in the context of didactic applications.

The single frequency measurements could be improved by implementing functionality for phase cycling and composite pulses.

Furthermore it would be very interesting to implement automation of pulse sequences. This means that certain parameters could be automatically varied between measurements, allowing for more complex sequences like Inversion Recovery (IR) sequences for measurement of the T1 time.

4.6. Simulation

The implementation of the simulator showcased the flexibility of the NQRduck software. Because it was implemented as a submodule of the spectrometer module, all the functionality of the spectrometer module, like automatic generation of a settings view, was available. This allowed for integration of the simulator, after translating the original work to Python, in a matter of hours.

For didactic applications, this module is very useful. It can be used to showcase the difference between the signals obtained from simulations and real-world experiments, as is already evident from the results presented in this thesis (Figure 3.27 & 3.31). These differences can be easily explained when looking at the full spectrum of the measurements (Figure D.2 & D.4).

For the LimeNQR measurements, a larger number of noise sources can be observed, likely from radio stations. Furthermore, the LimeNQR records the signal at an Intermediate frequency (IF) of 5 MHz, while the simulator simulates the signal at baseband (0Hz). Finally, the simulator is only as good as its input

parameters, meaning that these have to be carefully adjusted to represent the real-world conditions.

The simulator could also be used to plan pulse sequences for unknown samples in research projects.

For future work, the simulator should be thoroughly tested and verified on a wide range of different samples and temperatures. Furthermore, the simulator could easily be adapted for usage with NMR experiments.

4.7. Broadband Measurements

Broadband measurements with electrically and mechanically tuned probe coils work as evident from Figure 3.35 & 3.36. There are however limitations: First, because of the limitations in S_{11} measurements, the best reliably achievable S_{11} value for mechanically tuned probe coils is -25dB. For electrically tuned probe coils, it is slightly better at -30dB. For very high sensitivity measurements there is potential for further optimization.

The involvement of mechanically moving components and long measurements is error-prone. Here, the system would greatly benefit from automatic error detection and thorough testing of the system under a wide variety of experimental setups.

Furthermore, because the system is not relying on usage of a high-performance VNA, the measurement times are longer compared to previous implementations of the broadband system.

5. Conclusion

A modular framework for didactic NQR and NMR experiments was successfully developed. All goals, including the optional secondary goals were achieved.

The NQRduck program and its modules allow for easy programming of pulse sequences, data processing and exporting of said data. Furthermore, modules can be easily removed and added to the Core, allowing the program to be customized to specific requirements. Broadband measurements can be performed with electrically and mechanically tuned probe coils using the developed system for automatic Tuning and Matching.

The overall NQRduck system is significantly cheaper than commercial solutions and its open-source nature should allow for easy reproducibilty. The GUI could be improved by extensive user testing.

The modular nature allows for a wide range of potential improvements to the existing system. For future work, additional SDR-based spectrometers could be integrated into the existing framework. One example would be the STEMlab 125-14 (*Red Pitaya, Solkan, Slovenia*), which has previously been adapted for NMR experiments at the Institute of Biomedical Imaging.

Additionally, other magnetic resonance techniques like electron spin resonances (ESR) could be implemented with the existing hardware and then integrated into the existing framework.

A mobile educational kit which includes all components needed for magnetic resonance experiments could be realized by utilizing small portable power amplifiers and Hallbach arrays. This could allow for live demonstration of magnetic resonance experiments during lectures.

Bibliography

[1] C. Kittel, *Introduction to solid state physics*, 8. ed., [repr.] Hoboken, NJ: Wiley, 2013, 680 pp., ISBN: 9780471415268 (cit. on pp. 1, 5–7, 11).

[2] B. H. Suits, "NUCLEAR QUADRUPOLE RESONANCE SPECTROSCOPY," in *Handbook of Applied Solid State Spectroscopy*, Springer US, pp. 65–96. DOI: 10.1007/0-387-37590-2_2 (cit. on pp. 1, 11).

[3] C. Gösweiner, P. Lantto, R. Fischer, *et al.*, "Tuning nuclear quadrupole resonance: A novel approach for the design of frequency-selective MRI contrast agents," *Physical Review X*, vol. 8, no. 2, p. 021 076, Jun. 2018. DOI: 10.1103/physrevx.8.021076 (cit. on p. 1).

[4] H. Scharfetter, M. Bödenler, and D. Narnhofer, "A cryostatic, fast scanning, wideband NQR spectrometer for the VHF range," *Journal of Magnetic Resonance*, vol. 286, pp. 148–157, Jan. 2018. DOI: 10.1016/j.jmr.2017.12.004 (cit. on pp. 1, 22, 75).

[5] A. Doll, "Pulsed and continuous-wave magnetic resonance spectroscopy using a low-cost software-defined radio," *AIP Advances*, vol. 9, no. 11, Nov. 2019. DOI: 10.1063/1.5127746 (cit. on pp. 1, 19, 62).

[6] J. D. C. Franco, *A c++ api for the implementation ofsoftware-defined-radio nmr spectrometers*, Masters Thesis, Nov. 2019 (cit. on pp. 1, 2).

[7] C. A. Michal, "A low-cost multi-channel software-defined radio-based NMR spectrometer and ultra-affordable digital pulse programmer," *Concepts in Magnetic Resonance Part B: Magnetic Resonance Engineering*, vol. 48B, no. 3, Jul. 2018. DOI: 10.1002/cmr.b.21401 (cit. on pp. 1, 19).

[8] C. J. Hasselwander, Z. Cao, and W. A. Grissom, "Gr-mri: A software package for magnetic resonance imaging using software defined radios," *Journal of Magnetic Resonance*, vol. 270, pp. 47–55, Sep. 2016, ISSN: 1090-7807. DOI: 10.1016/j.jmr.2016.06.023 (cit. on p. 1).

[9] L. Kaltenleitner, *Software defined radio based nuclear quadrupole resonance spectrometer*, Masters Thesis, Institute of Biomedical Imaging, 2022 (cit. on pp. 1, 19, 25, 26, 42, 78, 82, 110).

[10] I. T. Union, *Recommendation itu-r v.431-8: Nomenclature of the frequency and wavelength bands used in telecommunications*, International Telecommunication Union, Aug. 2015 (cit. on p. 2).

[11] M. H. Levitt, *Spin dynamics, Basics of nuclear magnetic resonance*, 2. ed., repr. Chichester: Wiley, 2011, 714 pp., Includes bibliographical references, ISBN: 9780470511176 (cit. on pp. 5–9).

[12] A. Abragam, *The principles of nuclear magnetism* (The @international series of monographs on physics), Repr. Oxford [u.a.]: Clarendon Pr., 1986, 599 pp., ISBN: 019852014X (cit. on p. 5).

[13] K. L. Sauer, C. A. Klug, J. B. Miller, and A. N. Garroway, "Using quaternions to design composite pulses for spin-1 nqr," *Applied Magnetic Resonance*, vol. 25, no. 3–4, pp. 485–500, Sep. 2004, ISSN: 1613-7507. DOI: 10.1007/bf03166543 (cit. on p. 13).

[14] D. J. Lurie, "Numerical design of composite radiofrequency pulses," *Journal of Magnetic Resonance (1969)*, vol. 70, no. 1, pp. 11–20, Oct. 1986, ISSN: 0022-2364. DOI: 10.1016/0022-2364(86)90359-8 (cit. on p. 13).

[15] M. Jouda, S. M. T. Delgado, M. A. Jouzdani, D. Mager, and J. G. Korvink, "ArduiTaM: Accurate and inexpensive NMR auto tune and match system," *Magnetic Resonance*, vol. 1, no. 1, pp. 105–113, Jun. 2020. DOI: 10.5194/mr-1-105-2020 (cit. on pp. 14, 15, 25).

[16] Z. Peterson, *S11 parameter vs. return loss vs. reflection coefficient: When are they the same?* Nov. 2020. [Online]. Available: https://resources.altium.com/p/s11-vs-return-loss-vs-reflection-coefficient-when-are-they-same (visited on 12/22/2023) (cit. on p. 14).

[17] D. Rytting, "Network analyzer error models and calibration methods," *White Paper, September*, 1998 (cit. on pp. 15, 16).

[18] D. Freude, *Quadrupolar nuclei in solid-state nuclear magnetic resonance*, Oct. 2000. DOI: 10.1002/9780470027318.a6112 (cit. on p. 17).

[19] *Limesdr, product page*. [Online]. Available: https://www.crowdsupply.com/lime-micro/limesdr (visited on 11/28/2023) (cit. on p. 19).

[20] A. Doll, *Limr: Nmr/epr with the limesdr*, Sourceforge, 09, Feb. 2022. [Online]. Available: https://sourceforge.net/projects/limr/ (visited on 11/28/2023) (cit. on p. 19).

[21] R. Caverly and G. Hiller, "Establishing the minimum reverse bias for a p-i-n diode in a high-power switch," *IEEE Transactions on Microwave Theory and Techniques*, vol. 38, no. 12, pp. 1938–1943, 1990, ISSN: 0018-9480. DOI: 10.1109/22.64577 (cit. on p. 22).

[22] M. P. J. Tiggelman, K. Reimann, F. Van Rijs, J. Schmitz, and R. J. E. Hueting, "On the trade-off between quality factor and tuning ratio in tunable high-frequency capacitors," *IEEE Transactions on Electron Devices*, vol. 56, no. 9, pp. 2128–2136, Sep. 2009, ISSN: 0018-9383. DOI: 10.1109/ted.2009.2026391 (cit. on p. 22).

[23] H. Scharfetter, "An electronically tuned wideband probehead for nqr spectroscopy in the vhf range," *Journal of Magnetic Resonance*, vol. 271, pp. 90–98, Oct. 2016, ISSN: 1090-7807. DOI: 10.1016/j.jmr.2016.08.008 (cit. on p. 23).

[24] J. Pfitzer, *Stepper-driven automatic tuning andmatching system for a nuclear-quadrupole resonance spectrometer*, Bachelors Thesis, Institute of Biomedical Imaging, Apr. 2022 (cit. on pp. 23, 27, 30–34, 36, 37, 43, 110).

[25] B. Nißl, *Snr optimization for highly sensitive wide-band nqr measurements in cryogenic environments*, Masters Thesis, Institute of Biomedical Imaging, 2022 (cit. on pp. 24, 68, 134).

[26] V. Gonzalez-Posadas, D. Castro-Galan, J. Jimenez-Martin, D. Segovia-Vargas, and C. Martin-Pascual, "Lumped high-low pass balun for ultra wide band printed balanced antennas," in *2006 First European Conference on Antennas and Propagation*, IEEE, Nov. 2006. DOI: 10.1109/eucap.2006.4584593 (cit. on p. 24).

[27] L. Friedrich, "Dokumentation der logikschaltung für einen rx/tx-switch," Bioimaging and Bioinstrumentation Project, Tech. Rep., Graz University of Technology, Institute of Biomedical Imaging, 2015 (cit. on p. 26).

[28] *Adf4351*, Datasheet, Rev. A, Analog Devices Inc., May 2012. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/adf4351.pdf (visited on 11/23/2023) (cit. on p. 32).

[29] D. Fanning, *Adf4351*, GitHub Repository, commit:93948f57a60802cea3327209a57c94b979e14975, 2021. [Online]. Available: https://github.com/dfannin/adf4351 (visited on 01/09/2024) (cit. on p. 32).

[30] *Hmc241qs16*, Datasheet, v04.0404, Analog Devices Inc. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/hmc241qs16.pdf (visited on 11/23/2023) (cit. on p. 32).

[31] *Hmc849*, Datasheet, v01.0818, Analog Devices Inc. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/hmc849a.pdf (visited on 11/23/2023) (cit. on p. 33).

[32] *Adac click, product page*. [Online]. Available: https://www.mikroe.com/adac-click-click (visited on 11/24/2023) (cit. on p. 34).

[33] *Ad5593r*, Datasheet, Rev. A, Analog Devices Inc., Aug. 2023. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad5593r.pdf (visited on 11/23/2023) (cit. on p. 34).

[34] L. Janavicius, *Ad5593r-library-arduino-esp32*, GitHub Repository, commit:fa2dd3122910ccfce4d646fde9e1ea95bf39aea9, 2021. [Online]. Available: https://github.com/LukasJanavicius/AD5593R-Arduino-ESP32-Library (visited on 11/20/2023) (cit. on p. 34).

[35] *Ad8302*, Datasheet, Rev. B, Analog Devices Inc., 2008. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/ad8302.pdf (visited on 11/23/2023) (cit. on p. 35).

[36] J. Johnson, "Design of a low cost vector network analyser," Master's thesis, College of Engineering and Computer Science, Australian National University, Oct. 2019 (cit. on p. 35).

[37] *Tmc2130*, Datasheet, Rev. 1.16, Analog Devices Inc., Mar. 2023. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/TMC2130_datasheet_rev1.16.pdf (visited on 11/22/2023) (cit. on p. 36).

[38] J. Loeliger and M. McCullough, *Version Control with Git Powerful Tools and Techniques for Collaborative Software Development, Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, Incorporated, 2012, p. 456, ISBN: 9781449345044 (cit. on p. 44).

[39] E. Gamma, Ed., *Design patterns, Elements of reusable object-oriented software* (Addison-Wesley professional computing series), 39. printing. Boston, Mass.: Addison-Wesley, 2011, 395 pp., Literaturverz. S. 375 - 381, ISBN: 9780201633610 (cit. on p. 44).

[40] WikipediaContributors, *"model–view–controller — Wikipedia, the free encyclopedia"*, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=1181180757 (visited on 11/23/2023) (cit. on p. 44).

[41] *Python glossar:term-duck-typing*. [Online]. Available: https://docs.python.org/3/glossary.html#term-duck-typing (visited on 11/29/2023) (cit. on p. 49).

[42] S. Collins, *A deeper look at signals and slots*, Website, Dec. 2005. [Online]. Available: https://web.archive.org/web/20070703100120/http://scottcollins.net/articles/a-deeper-look-at-signals-and-slots.html (visited on 12/07/2023) (cit. on p. 50).

[43] C. Graf, A. Rund, C. S. Aigner, and R. Stollberger, "Accuracy and performance analysis for bloch and bloch-mcconnell simulation methods," *Journal of Magnetic Resonance*, vol. 329, p. 107 011, Aug. 2021, ISSN: 1090-7807. DOI: 10.1016/j.jmr.2021.107011 (cit. on p. 65).

[44] C. Graf, *Simulation of bloch and bloch mcconnell equations using symmetric and asymmetric operator splitting methods*, GitHub Repository, commit:4f0a1f21d321f6e6fa925cacee49990c49487e0a, 2021. [Online]. Available: https://github.com/GrafChristina/BlochSim (visited on 11/28/2023) (cit. on p. 65).

[45] S. Wunderl, *Scout control app: Quickstart guide and technicaldocumentation*, Jun. 2021 (cit. on p. 68).

Appendix

Appendix A.

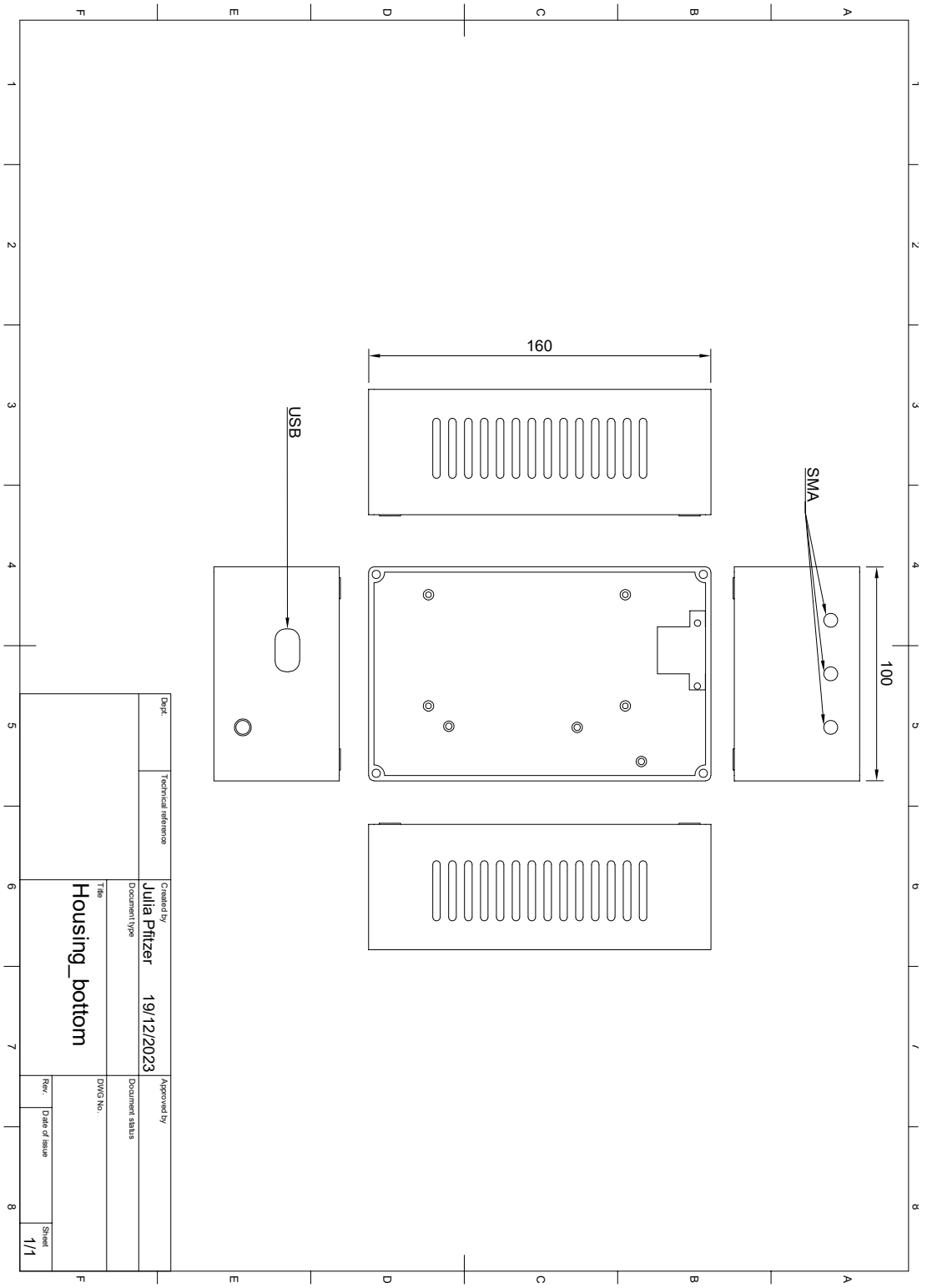
Appendix System Instrumentation

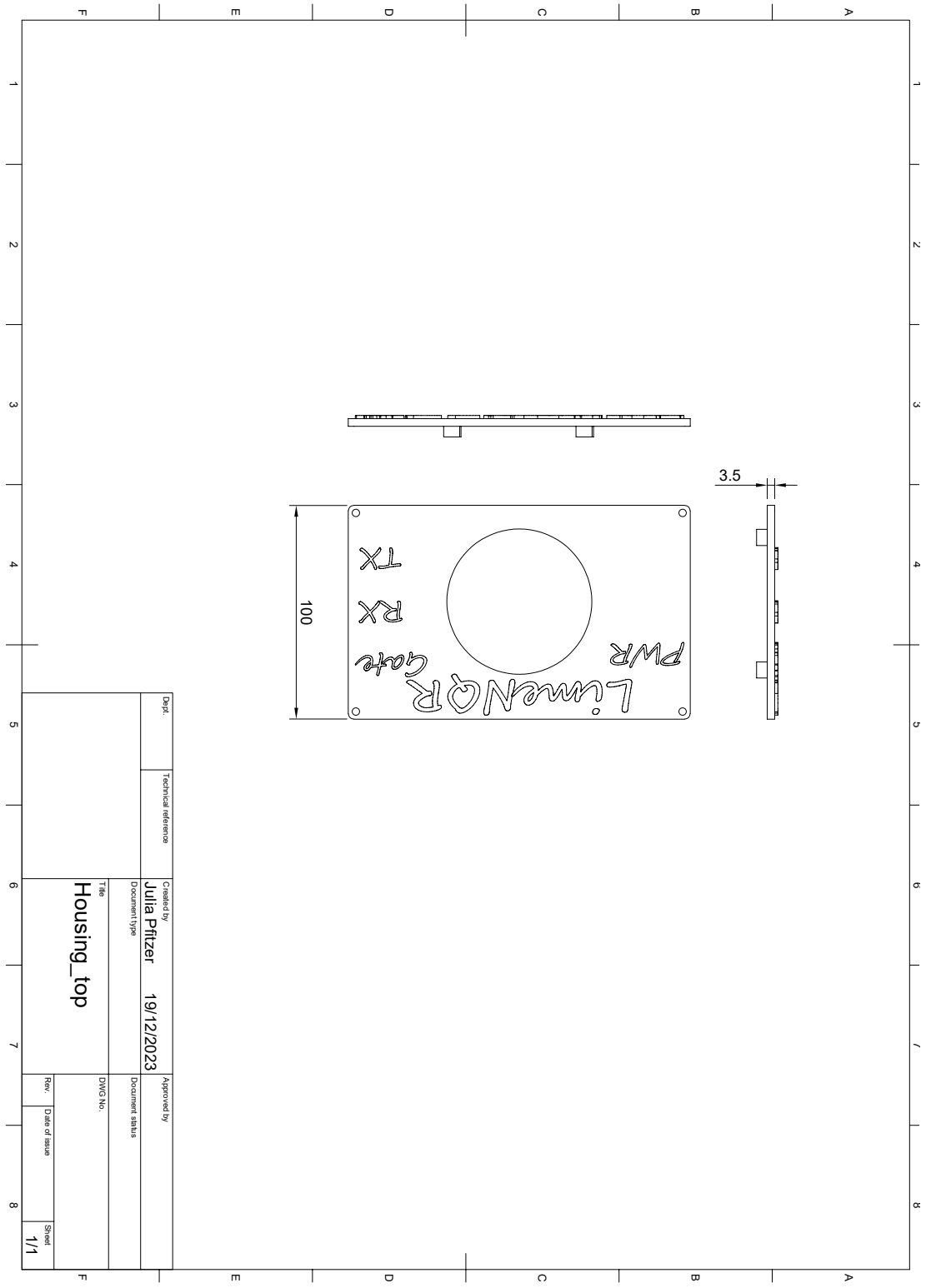
A.1. GPIO Table ATMV2

Table A.1.: ESP32 GPIO Assignments

Description	Pin Name	GPIO Number
SPI Clock	SCLK_PIN	14
SPI Master Out Slave In	MOSI_PIN	13
SPI Master In Slave Out	MISO_PIN	12
I2C Data	I2C_SDA	21
I2C Clock	I2C_SCL	16
ADF Latch Enable	LE_PIN	27
ADF Chip Enable	CE_PIN	25
M1 Enable	EN_PIN_M1	26
M1 Direction	DIR_PIN_M1	32
M1 Step	STEP_PIN_M1	33
M1 Chip Select	CS_PIN_M1	25
M1 Diag	DIAG1_PIN_M1	4
M2 Enable	EN_PIN_M2	17
M2 Direction	DIR_PIN_M2	19
M2 Step	STEP_PIN_M2	18
M2 Chip Select	CS_PIN_M2	5
M2 Diag	DIAG1_PIN_M2	2
Filter Switch A	FILTER_SWITCH_A	23
Filter Switch B	FILTER_SWITCH_B	22
RF Switch	RF_SWITCH_PIN	15

A.2. Housing LimeNQR


160
USB
SMA
100
Dept.
Technical reference
Created by
Julia Pfitzer
19/12/2023
Document type
Approved by
Document status
Title
Housing_bottom
DWG No.
Rev.
Date of issue
Sheet
1/1


3.5
100
TX
RX
PWR
LimeNQR
Cache
Dept.
Technical reference
Created by
Julia Pfitzer
19/12/2023
Document type
Approved by
Document status
Title
Housing_top
DWG No.
Rev.
Date of issue
Sheet
1/1

Appendix B.

Appendix Software Architecture

B.1. Installation NQRduck

At first, a new virtual environment is created.

```
# Create a new virtual environment named 'duck-venv'
python -m venv duck-venv

# Activate the virtual environment on Unix or MacOS
source duck-venv/bin/activate

# Activate the virtual environment on Windows
duck-venv\Scripts\activate.bat
```

Listing B.1: Creating and activating a virtual environment in Python.

Clone the `nqrduck` repository[1]. To install the package along with all dependencies and all modules, use the following command:

```
# Clone the repository
git clone https://github.com/nqrduck/nqrduck.git

# Navigate to the repository directory
cd nqrduck

# Install the package and all dependencies
pip install .[all]
```

[1] https://github.com/nqrduck/nqrduck

B.2. Development of NQRduck modules

Here, an exemplary workflow for the development of new NQRduck modules is given.

If one wants to develop a new module, a template is provided [2]. This template can be used as a starting point for a new module.

For example, if one wants to develop a new module called *myduck*, one would first clone the repository.

The template repository has the following file structure:

```
nqrduck-module/
|-- .gitignore
|-- LICENCE
|-- README.md
|-- pyproject.toml
|-- src/
|   |-- nqrduck_module/
|   |   |-- __init__.py
|   |   |-- controller.py
|   |   |-- model.py
|   |   |-- module.py
|   |   |-- resources/
|   |   |    |-- module.ini
|   |   |    |-- module_widget.ui
|   |   |-- view.py
|   |   |-- widget.py
```

Figure B.1.: File structure of the NQRDuck module Git repository.

[2] https://github.com/nqrduck/nqrduck-module

If one would now like to create a module with the example name *myduck*, the folder and file names would be modified in the following way:

```
nqrduck-myduck/
|-- .gitignore
|-- LICENCE
|-- README.md
|-- pyproject.toml
|-- src/
|   |-- nqrduck_myduck/
|   |   |-- __init__.py
|   |   |-- controller.py
|   |   |-- model.py
|   |   |-- myduck.py
|   |   |-- resources/
|   |   |    |-- module.ini
|   |   |    |-- myduck_widget.ui
|   |   |-- view.py
|   |   |-- widget.py
```

Figure B.2.: File structure of the newly created *myduck*

The class names inside the model, view and controller classes should also be modified to have a more representative name (e.g. MyDuckController, MyDuckView, MyDuckModel). The object created in the myduck file should also be renamed:

```python
from nqrduck.module.module import Module
from .model import MyDuckModel
from .view import MyDuckView
from .controller import MyDuckController

MyDuck = Module(MyDuckModel, MyDuckView, MyDuckController)
```

Then the .ini file inside the resources folder should be modified to represent the *myduck* module.

```
[META]
name = nqrduck-myduck
category = MyOwnModules
toolbar_name = MyDuck
tooltip = Template for a new module
```

Figure B.3.: Example for the modified .ini file. The category, toolbar name and tooltip can be freely chosen.

Additionally the pyproject.toml has to be modified to represent the new module structure:

```
...

[project]
name = "nqrduck-myduck"
version = "0.0.1"
authors = [
  { name="YOUR NAME", email="your@e.mail" },
]

description = Your description"

...

dependencies = [
    "nqrduck",
    "pyqt6",
    ...
]

[project.entry-points."nqrduck"]
"nqrduck-myduck" = "nqrduck_myduck.myduck:MyDuck"
```

Listing B.2: modified contents of pyproject.toml

Now functionality can be implemented. For example the *myduck_widget.ui* could be modified using Qt Designer. The widget.py file can then be updated using *pyuic6*.

```
# Navigate to the repository directory
cd nqrduck-myduck

# Generate your widget.py
pyuic6 src/nqrduck_myduck/resources/myduck_widget.ui > src/
    nqrduck_myduck/widget.py
```

The module should be installed to test its functionality. Ideally, the module is installed with the -e argument of pip to install it in editable mode. This means changes in the source code of the project will be applied without reinstalling the module.

```
# Navigate to the repository directory
cd nqrduck-myduck

# Install the package and all dependencies
pip install -e .
```

B.3. Example pyproject.toml

```
[build-system]
requires = ["hatchling"]
build-backend = "hatchling.build"

[project]
name = "nqrduck-module"
version = "0.0.1"
authors = [
  { name="Author Name", email="author@e.mail" },
]

description = "A template for nqrduck modules."
readme = "README.md"
license = { file="LICENSE" }
requires-python = ">=3.8"

classifiers = [
    "Programming Language :: Python :: 3",
    "License :: OSI Approved :: MIT License",
    "Operating System :: OS Independent",
]

dependencies = [
    "matplotlib",
    "pyqt6",
    "nqrduck",
]

[project.entry-points."nqrduck"]
"nqrduck-module" = "nqrduck_module.module:module"
```

Listing B.3: contents of pyproject.toml

B.4. NQRduck File Extensions

Table B.1.: File Extensions and Module Descriptions

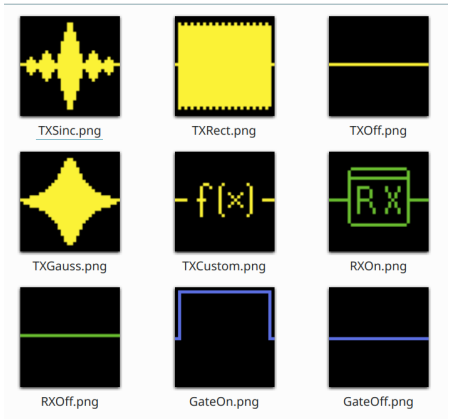
Extension	**Module**	**Description**
.s11	nqrduck-autotm	Contains magnitude and phase data from frequency sweep experiments.
.cal	nqrduck-autotm	Stores calibration data for the short, open, and load (SOL) calibration process.
.pos	nqrduck-autotm	Records saved stepper positions for various frequencies utilized in the probe tuning and matching process.
.meas	nqrduck-measurement	Holds data from single-frequency experiments, encapsulated in a Measurement object.
.broad	nqrduck-broadband	Holds data from multiple Measurement objects, providing a comprehensive overview of broadband experiment results.
.quack	nqrduck-spectrometer	Represents a pulse sequence file, defining the sequence of pulses executed in an NMR or NQR experiment.

B.5. Sprites

a) Duck Kick Animation

b) Lab Duck Animation

c) Logos



d) Pulse Parameters

Figure B.4.: Logo of the NQRduck program.

Appendix C.

Appendix Modules

C.1. AutoTM Module

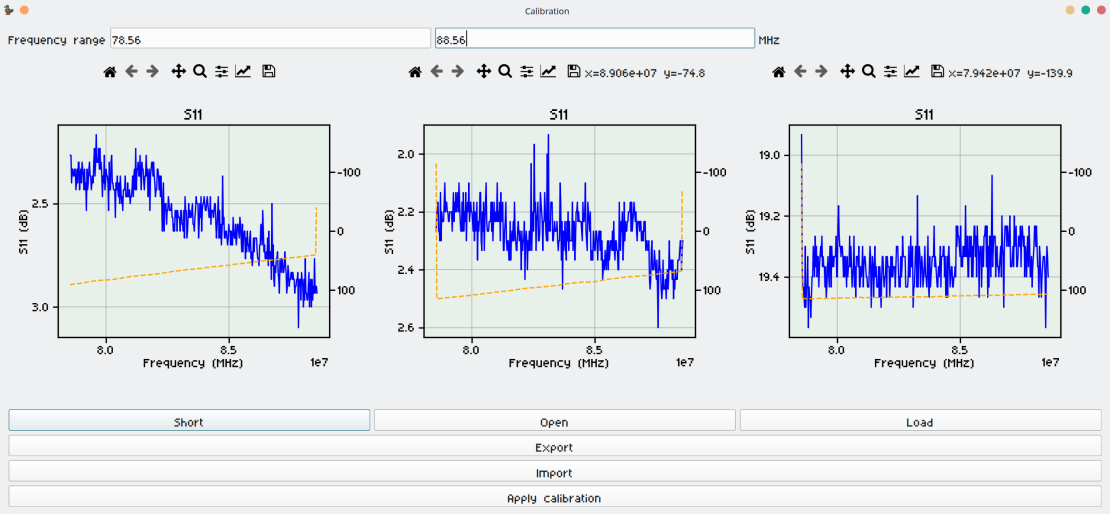


Figure C.1.: The screenshot illustrates the calibration interface. Calibration proceeds by setting the desired frequency range within the calibration window. Begin with the short calibration: attach the short adapter to the reference plane (for instance, where the cable connects to the probe coil) and click the 'Short' button. Follow the same steps for open calibration, attaching the open adapter and then pressing the 'Open' button. The load calibration is conducted similarly. After completion, the calibration data can be saved as a .cal file. To activate the calibration, click the 'Apply Calibration' button.

C.2. Simulator

The step-by-step calculation for the reference voltage is as follows:
First, calculate the magnetization of the sample:

$$|M| = \gamma \cdot \frac{2 \cdot N_{\text{atoms}}}{2I+1} \cdot \frac{h^2 \cdot \nu_{\text{res}}}{k \cdot T} \cdot S \tag{C.1}$$

Then, calculate the cross-sectional area of the coil:

$$A_{\text{coil}} = \pi \left(\frac{d_{\text{coil}}}{2} \right)^2 \tag{C.2}$$

Finally, calculate the reference voltage using the previously calculated values:

$$|V_{\text{ref}}| = N \cdot A_{\text{coil}} \cdot \mu_0 \cdot M \cdot \nu_{\text{res}} \cdot P \cdot F \tag{C.3}$$

Here, the terms represent the following parameters:

- M: Magnetization of the sample in A/m
- N: Number of turns in the coil
- A_{coil}: Cross-sectional area of the coil in m^2
- μ_0: Permeability of free space ($4\pi \times 10^{-7} \frac{H}{m}$)
- γ: Gyromagnetic ratio of the sample in $\frac{MHz}{T}$
- N_{atoms}: Number of atoms per sample $\frac{1}{m^3}$
- I: Nuclear angular momentum quantum number
- h: Planck's constant in $J \cdot s$
- ν_{res}: Resonant frequency of the sample in MHz
- k: Boltzmann constant in $\frac{J}{K}$
- T: Temperature of the sample in K
- S: Spin angular momentum in $J \cdot s$
- d_{coil}: Diameter of the coil in m
- P: Powder factor of the sample
- F: Filling factor of the sample

These equations can also be found in [25].

C.3. Pulse Programmer JSON


```
{
    "name": "Untitled pulse sequence",
    "events": [
        {
            "name": "pulse",
            "duration": "0.000003000000",
            "parameters": [
                {
                    "name": "TX",
                    "value": [
                        {
                            "name": "Relative TX Amplitude",
                            "value": 1.0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Phase",
                            "value": 0.0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Pulse Shape",
                            "value": {
                                "name": "Rectangular",
                                "parameters": [],
                                "expression": "1",
                                "resolution": "3.255208E-8",
                                "start_x": -1.0,
                                "end_x": 1.0
                            },
                            "type": "Function"
                        }
                    ]
                },
                {
                    "name": "RX",
                    "value": [
                        {
                            "name": "RX",
                            "value": false,
                            "type": "Boolean"
                        }
                    ]
                }
            ]
        },
        {
            "name": "blank",
            "duration": "0.000007",
            "parameters": [
                {
                    "name": "TX",
                    "value": [
                        {
                            "name": "Relative TX Amplitude",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Phase",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Pulse Shape",
                            "value": {
                                "name": "Rectangular",
                                "parameters": [],
                                "expression": "1",
                                "resolution": "3.255208E-8",
                                "start_x": -1.0,
                                "end_x": 1.0
                            },
                            "type": "Function"
                        }
                    ]
                },
                {
                    "name": "RX",
                    "value": [
                        {
                            "name": "RX",
                            "value": false,
                            "type": "Boolean"
                        }
                    ]
                }
            ]
        },
```

```
        {
            "name": "rd",
            "duration": "0.000020000000",
            "parameters": [
                {
                    "name": "TX",
                    "value": [
                        {
                            "name": "Relative TX Amplitude",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Phase",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Pulse Shape",
                            "value": {
                                "name": "Rectangular",
                                "parameters": [],
                                "expression": "1",
                                "resolution": "3.255208E-8",
                                "start_x": -1.0,
                                "end_x": 1.0
                            },
                            "type": "Function"
                        }
                    ]
                },
                {
                    "name": "RX",
                    "value": [
                        {
                            "name": "RX",
                            "value": true,
                            "type": "Boolean"
                        }
                    ]
                }
            ]
        },
        {
            "name": "TR",
            "duration": "0.001000",
            "parameters": [
                {
                    "name": "TX",
                    "value": [
                        {
                            "name": "Relative TX Amplitude",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Phase",
                            "value": 0,
                            "type": "Numeric"
                        },
                        {
                            "name": "TX Pulse Shape",
                            "value": {
                                "name": "Rectangular",
                                "parameters": [],
                                "expression": "1",
                                "resolution": "3.255208E-8",
                                "start_x": -1.0,
                                "end_x": 1.0
                            },
                            "type": "Function"
                        }
                    ]
                },
                {
                    "name": "RX",
                    "value": [
                        {
                            "name": "RX",
                            "value": false,
                            "type": "Boolean"
                        }
                    ]
                }
            ]
        }
    ]
}
```

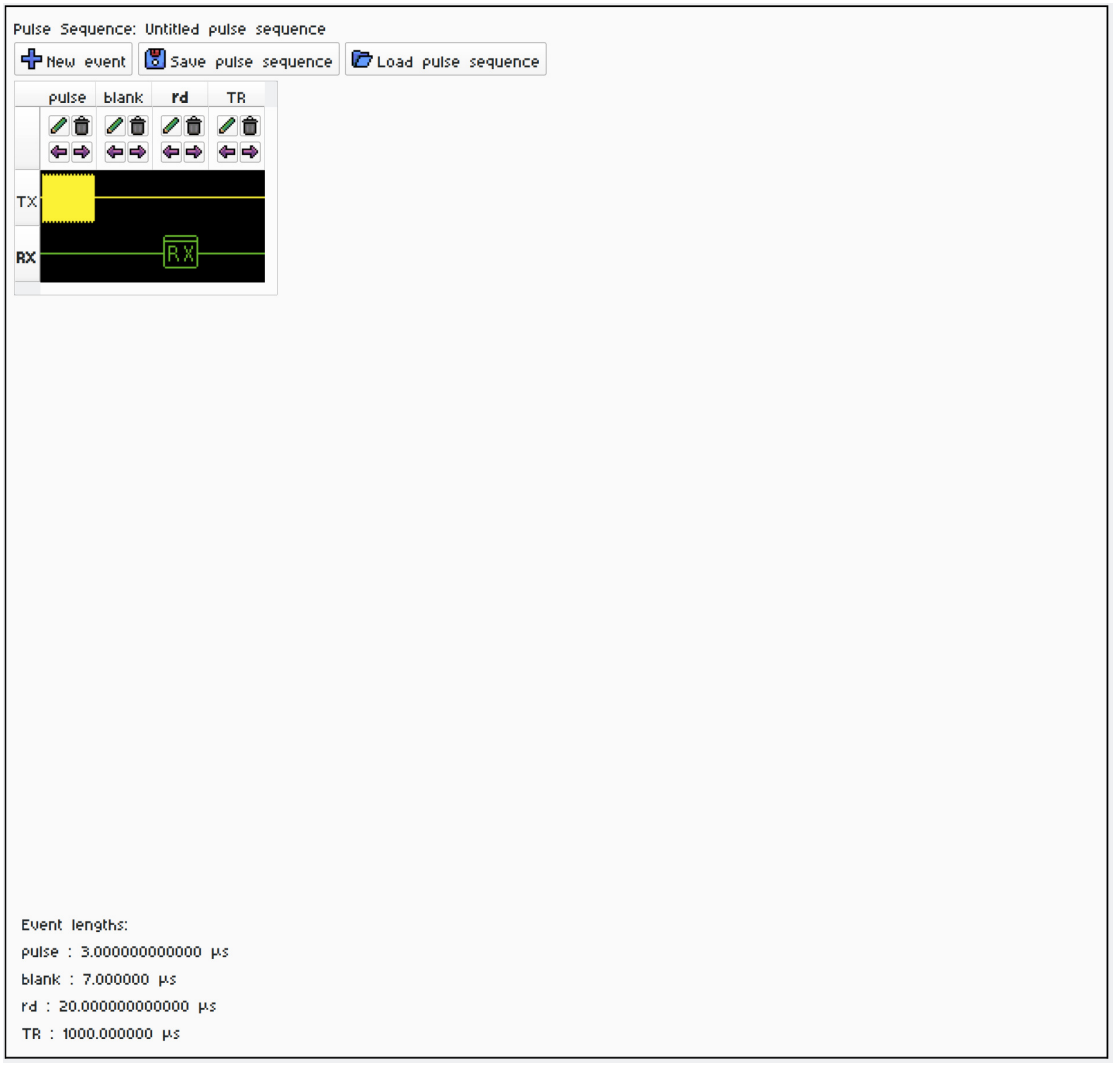


Figure C.2.: Example FID sequence visualized with the Pulse Programmer module. The visualization corresponds to the previously depicted JSON.

Appendix D.

Appendix Results

D.1. LimeSDR Pulse Shaping

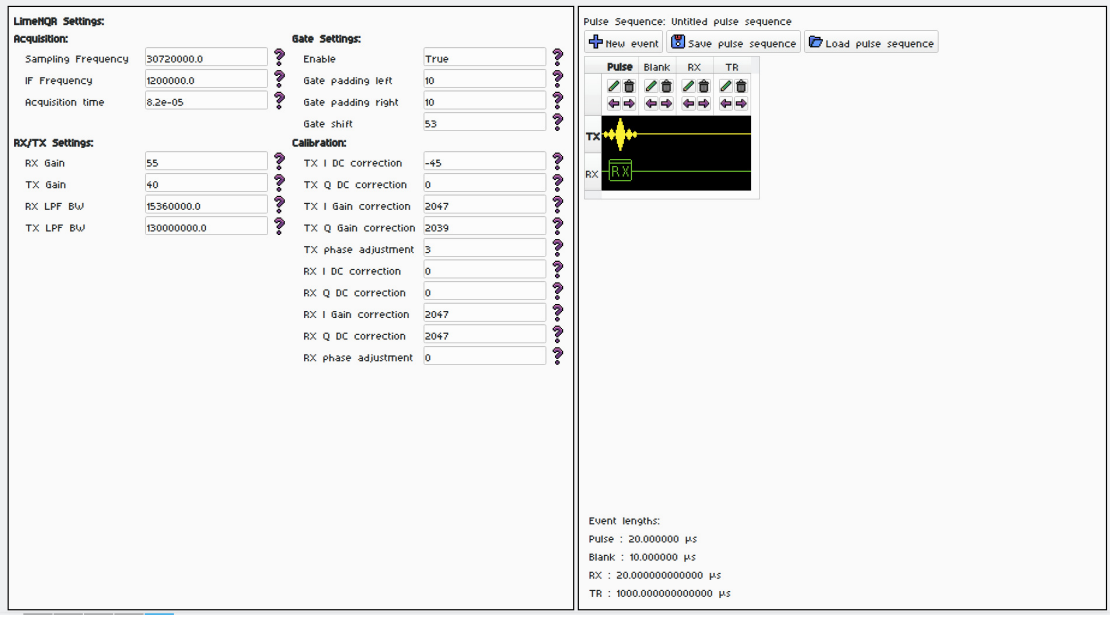


Figure D.1.: Pulse sequence and settings used for the pulse shaping experiments.

D.2. Single Frequency Measurements

D.2.1. LimeNQR: FID

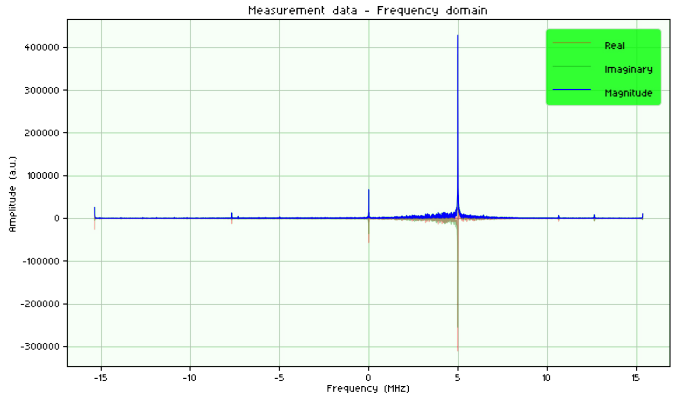


Figure D.2.: Full FID Spectrum of the $BiPh_3$ sample. The pulse was recorded at an IF of 5MHz. No phase correction was applied to the data points.

D.2.2. LimeNQR: SE

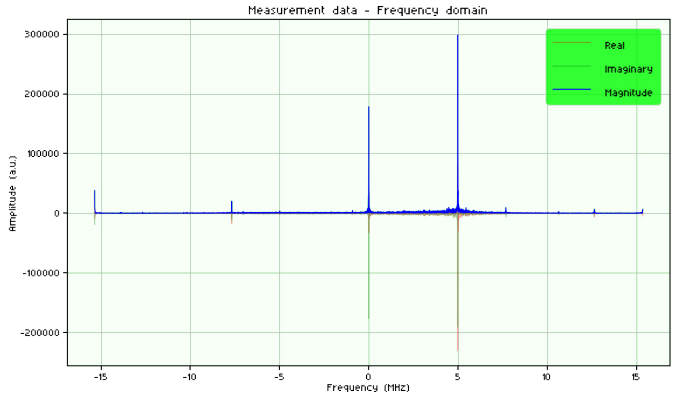


Figure D.3.: Full SE Spectrum of the $BiPh_3$ sample. The pulse was recorded at an IF of 5MHz. No phase correction was applied to the data points.

D.2.3. Simulator: FID

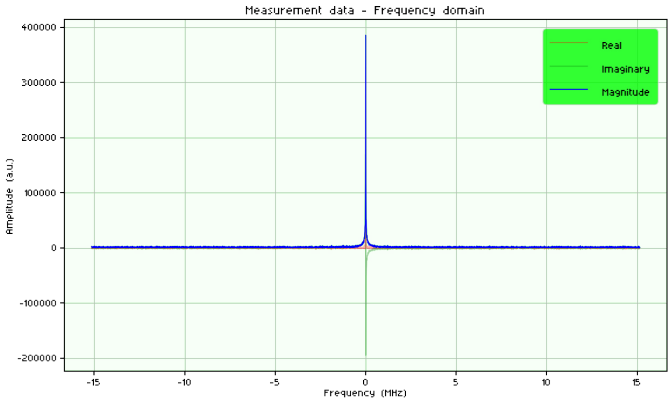


Figure D.4.: Full simulated FID spectrum of the $BiPh_3$ sample.

D.2.4. Simulator: SE

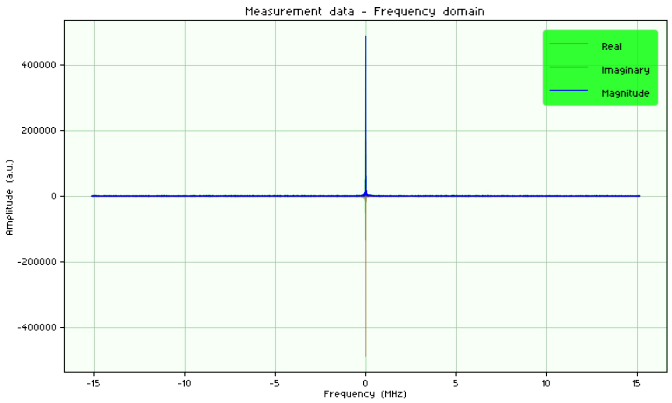


Figure D.5.: Full simulated SE Spectrum of the $BiPh_3$ sample.