ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Master's Thesis

21st December, 2016

Spectroscopy Automation and Sample Characterization of Superconducting Qubits

Quantum Device Lab, Department of Physics, ETH Zürich

Author:
Simon Storz

Supervisor:
Prof. Dr. A. Wallraff

Advisor:
Johannes Heinsoo

Abstract

Superconducting qubits are a promising candidate for the realization of a quantum computer, quantum networks and for experiments towards fundamental research. As the superconducting circuits used for the experiments in the circuit quantum electrodynamic (cQED) architecture become more complicated over the years, automized software for performing the experiments becomes inevitable. Here we develop a new software suit that allows for automized spectroscopy experiments with superconducting qubits. The present thesis describes this software in detail and explains how it can be extended for further use. Moreover the thesis provides an overview of the characterization experiments of a chip in the cQED architecture and suggests a recipe of how to determine the defining parameters of superconducting qubits in a systematic way.

Contents

1 Introduction **4**
- 1.1 Quantum Technologies . . . 4
- 1.2 Towards a Quantum Computer . . . 5
- 1.3 Motivation for this project . . . 6

2 Superconducting Qubits **7**
- 2.1 Introduction . . . 7
- 2.2 The Cooper Pair Box . . . 9
- 2.3 The Transmon Qubit . . . 10
- 2.4 Circuit QED . . . 11
- 2.5 Drive and Readout of a Transmon . . . 14
- 2.6 Coherence . . . 15

3 Experiment setup **17**
- 3.1 Overview . . . 17
- 3.2 Cryogenics . . . 17
- 3.3 Signal modulation . . . 20
- 3.4 Downconversion . . . 21
- 3.5 Synchronization . . . 21

4 Software **24**
- 4.1 SweepSpot framework . . . 24
  - 4.1.1 Concept . . . 24
  - 4.1.2 Definition of an experiment . . . 25
  - 4.1.3 SweepSpot generator . . . 31
  - 4.1.4 SweepSpot.vi . . . 33
  - 4.1.5 Save measurement data / file handling . . . 37
- 4.2 SweepSpot frontend . . . 40
- 4.3 Efficiency of time and memory usage . . . 42
- 4.4 Improvements and differences to Cleansweep . . . 44

5 Spectroscopy Automation **45**
- 5.1 Software overview . . . 45
- 5.2 Spectroscopy.vi . . . 46
- 5.3 Resonator/Qubit spectroscopy . . . 47

5.4 Combined Spectroscopy . . . 50
5.5 Parking qubits . . . 50
5.6 Tracking qubits . . . 51

6 Sample characterization **54**
6.1 Sample for example data . . . 54
6.2 Checking the sample . . . 55
6.2.1 Purcell filter spectroscopy . . . 55
6.2.2 Magnetic field dependence . . . 56
6.3 Qubit spectroscopy . . . 57
6.3.1 Finding the optimal readout frequency . . . 58
6.3.2 Broad and fine qubit spectroscopy . . . 58
6.3.3 AC Stark shift and pulsed spectroscopy . . . 60
6.3.4 Checking the behavior of the qubit . . . 60
6.3.5 Anharmonicity . . . 63
6.3.6 Exact Transition Frequency . . . 63
6.3.7 QScale calibration . . . 65
6.3.8 Coherence Times . . . 66
6.4 Tracking the qubit . . . 68
6.5 Further experiments . . . 69

7 Conclusion **70**
7.1 Summary . . . 70
7.2 Proposals . . . 70

Acknowledgments **72**

A Measurement configurations of time statistics experiments **73**

B Adding features to SweepSpot **74**
B.1 General remarks . . . 74
B.2 Adding a new sweep type . . . 75
B.3 Adding a new instrument . . . 76
B.4 Adding a new acquisition device . . . 77
B.5 Using SweepSpot as a module . . . 78

Chapter 1

Introduction

1.1 Quantum Technologies

In the last century quantum physics has revolutionized our understanding of nature and the technologies we interact with in our daily lives. Quantum theories originate in a paper of Max Planck in 1901 [40] solving the ultraviolet catastrophe problem by suggesting that radiation comes in quantized units. Based on that Albert Einstein in 1905 proposed the existence of the photon [19]. In the following years scientists discovered the substructures of the atom and developed a mathematical theory of the phenomenons at tiny scales. Especially notable are the discovery of the wave particle duality [16], Wolfgang Paulis exclusion principle [39], Erwin Schrödingers famous equation of the same name [44] and Werner Heisenbergs uncertainty relation [28].
But quantum physics has not only fundamentally changed how we think about nature, it has also changed technological, economical and sociological foundations of our civilization. Quantum physics gave birth to lasers that revolutionized the way humans communicate. Highly precise atomic clocks allowed us to precisely determine our position via GPS and with magnetic resonance imaging (MRI) [36] quantum physics even found its way to medicine. The greatest effect of quantum theory on our daily lives however was the development of the transistor which led to the invention of the computer. As these inventions, especially computers, have radically changed the world, the development of quantum theory in the early 20th century is often referred to as the ”first quantum revolution”.

In the second half of the 20th century scientists amongst other things began to investigate the phenomenon of entanglement. As the understanding of this unfamiliar feature of the theory grew, scientists became able to propose new applications of quantum physics that are based on entanglement. In 1991, Artur Ekert improved Charles Bennetts secure key distribution scheme [4] by using entangled quantum states [20]. Two years later Bennett proposed an algorithm to teleport a quantum state [5]. Alongside that physicists discovered that technology can make use of quantum physics to outperform classical computers in certain tasks. Most prominently, Peter Shor showed in 1994 [47] that so-called quantum computers could perform prime factorization in polynomial time instead of nearly exponential time for classical computers [50]. A little later Lov Grover showed that quantum computers are way more efficient than classical computers in searching a database [23]. The development of quantum technologies such as fast quantum computers alongside with secure communication, efficient simulation of other quantum processes

and highly accurate quantum sensors is believed to lead to a "second quantum revolution".

From all of these new technologies quantum computers could probably have the biggest influence on humanity. Quantum computers could be used to break most of today's public key algorithms [12], they could revolutionize economy through fast and powerful algorithms predicting changes on the stock market [21], they could be used to dramatically improve radiation therapy and drug development [30] and they could lead to a deeper understanding of nature itself thanks to simulations of other quantum mechanical processes [10]. But the exponential speedup a quantum computer could provide for certain calculations is only one of the main motivations to develop a quantum processor. The other reason is that classical chips soon cannot be made any smaller. Quantum effects that begin to interfere with the electronics form a fundamental limit for the development of better processors [38]. A quantum computer could solve this problem by instead of being limited by quantum effects making use of them.

1.2 Towards a Quantum Computer

The first specific attempts to control single quantum systems were done in the 1970s [38]. This was a necessary condition for the implementation of the first qubits. The first qubits were photons with their perpendicular polarization state as the computational basis. At these days, single photon sources got developed and the technology to detect photons was already evolved. However, as the photons only interact weakly with each other, two qubit gates are hard to implement. This is a serious drawback for an architecture for a quantum computer [38].
Another early implementation of qubits was achieved with spins of atomic nuclei. The so-called nuclear magnetic resonance (NMR) technology led to an early 3 qubit quantum computer [32]. But also NMR technology has serious drawbacks for quantum computing. First, as the nuclear magnetic moment is small, a large number of qubits must be used to be able to generate a detectable signal. Second, it is extremely hard to prepare the spins in a pure (ground) state at room temperature where the technology is usually used at, as the spin energy $\hbar\omega$ is significantly lower than $k_B T$ in this temperature regime [38].

Furthermore qubits have been realized with quantum dots [26] and trapped ions [37]. Trapped ions are considered to be one of the most promising candidates for quantum computing implementations, amongst other things as there exists a theoretical approach to scale up an ion trap quantum computer [34].

The other auspicious technology for quantum information processing are superconducting qubits. Superconducting qubits are artificial atoms coupled to artificial microwave cavities on a chip (typically on the order of 1cm^2; see Chapter 2). The interactions between the electromagnetic field (photons) and the qubits are described by the theory of circuit quantum electrodynamics (circuit QED) which is deeply connected to the evolved theory of cavity QED [49]. Superconducting qubits are especially interesting as their characteristics can be engineered to a large amount. For fabrication, standard lithography technique can be used which makes it convenient to manufacture the chips and is promising for the scalability of the technology.
The research done in the Quantum Device Lab (Qudev) at ETH is based on superconducting qubits. Therefore circuit QED is the general setting this Master's thesis is based on.

1.3 Motivation for this project

The qubits on the chips need to be tested and characterized during their development. In the long run quantum computers are expected to use a few hundreds or thousands of qubits. The more qubits there are on a sample and the more complex the design of the chip gets the more time-consuming it becomes to perform measurements by hand. Especially for the characterization of the samples automation becomes crucial.
As the samples are fabricated by the Qudev group itself they need to be tested and characterized before they can be used to perform quantum algorithms. Also in between subsequent experiments on a sample one often needs to do spectroscopy measurements again. The aim of this Master's project was to automize the most important spectroscopy experiments. In order to do this the measurement software used in the laboratory had to be rewritten. For that a new software suit called *SweepSpot* was developed. Its concept, its advantages and the procedure of sample characterization itself are the core subjects of this thesis.
The thesis is structured as follows: First, in Chapter 2 a general overview of superconducting qubits is provided. Then in Chapter 3 the setup of a typical quantum computing experiment is explained. Afterwards in Chapter 4 the new software suit is presented and Chapter 5 shows how the software is used to automize spectroscopy experiments. This is followed by a discussion in Chapter 6 about a systematic approach for sample characterization and about the scientific background of the experiments. Finally, the thesis will be summarized and proposals for further projects will be made.

Chapter 2

Superconducting Qubits

In 2000, Di Vincenzo summarized 5 criteria that are required for a physical system to be able to perform quantum computing algorithms [18]:

1. a scalable physical system with well characterized qubits;
2. the ability to initialize the state of the qubits to a simple fiducial state;
3. long relevant decoherence times, much longer than the gate operation time;
4. a "universal" set of quantum gates;
5. a qubit-specific measurement capability.

There are several types of superconducting qubits that fulfill this criteria. A general overview of the technology is provided in the first part of this chapter, followed by a discussion of one of the first superconducting qubits: the Cooper pair box. Then its successor, the Transmon qubit, will be described. Afterwards there will be given a short overview of circuit QED and it will be shown how the state of the Transmon qubit can be manipulated and read out. Finally the coherence times of the qubits will be discussed.

2.1 Introduction

Superconducting qubits are artificial atoms made up of superconducting metal. Their basic structure can be represented by an electric circuit diagram. The basic concept is best understood when first investigating a parallel LC oscillator, shown in Figure 2.1. Applying Kirchoff's law to the LC circuit leads to its equation of motion [3]

$$L\frac{d^2Q}{dt^2} + \frac{1}{C}Q = 0. \tag{2.1}$$

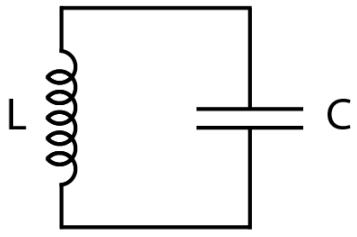


Figure 2.1: A simple LC circuit. Figure adapted from [24].

From here, the Lagrangian $\mathcal{L}(p, \dot{Q}) = (1/2)L(dQ/dt)^2 - (1/2C)Q^2$ can be derived and using a Legendre transformation $H(p,q) = p\dot{q} - \mathcal{L}$ the Hamiltonian of the system is found as

$$H(Q, \Phi) = \frac{Q^2}{2C} + \frac{\Phi^2}{2L}. \tag{2.2}$$

Here Φ denotes the flux which takes the role of the canonical momentum p whereas the charge Q represents the generalized position q.

The next step is to quantize the system. But first it has to be verified whether the LC circuit can actually be considered as being quantum. The answer is yes, if certain conditions are fulfilled. Generally, a system can be considered behaving quantum mechanically if its degrees of freedom are well decoupled from the environment and its state is close to the ground state of the system. Like that quantum decoherence, which transforms a quantum system to a classical one, can be suppressed. The collective degrees of freedom in an electric LC circuit are the charge Q stored on a capacitor and the flux Φ stored in an inductor [3]. The requirement is therefore to ensure the electric signals that pass from one part of the chip to another do not dissipate. This can be done by manufacturing the metal parts on the chip out of superconducting material [17]. Often Aluminium or Niobium are used for that [48].
Furthermore, thermal noise on the signals needs to be suppressed to a regime where $kT << \hbar\omega_{ge}$ where ω_{ge} is the transition frequency between the lowest two energy states of the qubit. As the typical transition frequency of a superconducting qubit lies in the gigahertz range (typically at 5-20 GHz) [17] the temperature of the chip must be in the millikelvin region (typically around 20 mK). Suppressing thermal noise and ensuring superconductivity therefore requires the sample to be cooled down using a cryostat (see Section 3.2).

Cooling the chip to the mK range therefore ensures that the qubit materials are in the superconducting regime and that the system behaves quantum mechanically. Now the Hamiltonian in Eq. 2.2 can be safely quantized. This is done by replacing the canonical conjugate variables of the system by their corresponding anticommuting quantum operators:

$$Q \rightarrow \hat{Q}, \Phi \rightarrow \hat{\Phi} \tag{2.3}$$

with $[\hat{Q}, \hat{\Phi}] = i\hbar$. Following the standard quantisation of a harmonic oscillator and using $\omega = \frac{1}{\sqrt{LC}}$, $\hat{Q} = i\sqrt{\hbar/2Z}(\hat{a} - \hat{a}^\dagger)$ and $\hat{\Phi} = \sqrt{Z\hbar/2}(\hat{a} + \hat{a}^\dagger)$, Eq. 2.2 becomes

$$\hat{H} = \hbar\omega(\hat{a}^\dagger\hat{a} + \frac{1}{2}). \tag{2.4}$$

This leads to a discrete energy spectrum with equally separated levels. However, as the aim is to construct a qubit, i.e. a two level system, one must find a way to suppress transitions to other levels than between the ground state and the first excited state. In particular, anharmonicity has to be introduced. The only known dissipation free electrical element that achieves that is a Josephson junction [3].

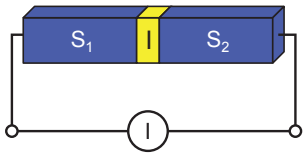


Figure 2.2: A drawing of a Josephson junction. The blue outer parts represent the superconducting conductors and the yellow colored part in the middle denotes the insulating film. Figure adapted from [3].

The Josephson Junction, first proposed by David Josephson in 1962 [33], contains two superconducting electrodes separated by a thin insulator, as shown in Figure 2.2. As the wavefunctions of the two electrodes overlap, Cooper pairs (electrons) can tunnel through the insulator from one side to the other.

The Josephson Junction can be characterized by two energy parameters: First, the Josephson energy

$$E_J = \frac{\Phi_0 I_C}{2C} \tag{2.5}$$

is the energy stored in the Josephson junction, I_C representing the current over the junction, C its capacitance and $\Phi_O = h/2e$ the flux quantum. Moreover, the charging energy

$$E_C = \frac{e^2}{2C} \tag{2.6}$$

denotes the energy needed to transfer one electron from one side of the Josephson junction to the other. These two energies, together with the phase difference δ across the element and the total number N of excess Cooper pairs on one electrode relative to the neutral state fully characterize the state of the Josephson junction [3].
There are several qubit designs that make use of the Josephson Junction. They can be classified as charge, flux or phase qubits corresponding to the quantity that determines the state of the qubit. One of the first designs was the Cooper pair box (a charge qubit), first proposed in 1987 by Markus Büttiker [11].

2.2 The Cooper Pair Box

The schematics of a Cooper pair box (CPB) in combination with a superconducting quantum interference device loop (SQUID) is shown in Figure 2.3. A SQUID is an electric loop which is highly sensitive to external magnetic fields. The SQUID splits the Josephson Junctions into two parts. The Cooper pair box further contains a so-called "reservoir" of electrons that is separated from an "island" by the two Josephson junctions. The SQUID is added to tune the Josephson energy of the qubit by applying a magnetic flux Φ.
Assuming the two Josephson junctions have equal Josephson energies the SQUID can be modeled as a single Josephson junction with effective energy

$$E_J(\Phi) = 2E_J^{max} |\cos(\frac{\pi\Phi}{\Phi_0})|. \tag{2.7}$$

The maximal Josephson energy of the SQUID is given by the sum of the individual Josephson energies, thus $E_J^{max} = 2E_J$. The Hamiltonian of the system, given by the potential (E_J) and kinetic energy (electrostatic energy of the capacitor), can be found as [3]

$$H_{CPB} = 4E_C(\hat{N} - N_G) - E_J(\Phi)\cos(\hat{\delta}). \tag{2.8}$$

$\hat{N}$ is the operator corresponding to the number of excess Cooper pairs on the island, $N_G = -C_G V_G/2e$ stands for the charge on the gate capacitor in units of $2e$ and $E_C = e^2/2C$ denotes the charging energy. The qubit is tunable via the magnetic flux that influences E_J and by the external gate voltage that changes N_G.

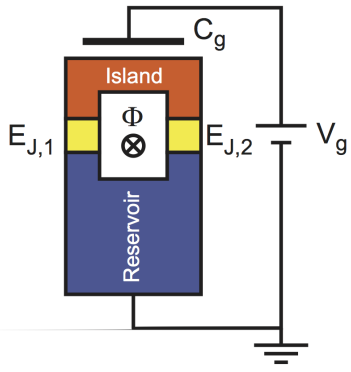


Figure 2.3: A scheme of the Cooper pair box with a SQUID loop. Figure adapted from [3].

The Hamiltonian in Eq. 2.8 can be rewritten in the charge representation where $\hat{N}|N\rangle = n|N\rangle$, $[\hat{\delta}, \hat{N}] = i$ and $e^{\pm i\hat{\delta}}|N\rangle = |N \mp 1\rangle$. In that basis it reads [9]

$$H_{CPB} = \sum_N [4E_C(N - N_G)^2 |N\rangle\langle N| - \frac{E_J}{2}(|N\rangle\langle N+1| + |N+1\rangle\langle N|)]. \tag{2.9}$$

This representation emphasizes that the number of Cooper pairs on the island can only be changed via E_J. The exact eigenenergies of the Cooper pair box can be expressed with the Mathieu functions [35]

$$E_m(N_G) = E_C a_{2[N_G + k(m,N_G)]}(-\frac{E_J}{2E_C}) \tag{2.10}$$

where $a_i(q)$ is the Mathieu's characteristic value and $k(m, N_G)$ is a function that sorts eigenvalues. Figure 2.4 shows the first three eigenenergies of the Cooper pair box in relation to the applied gate voltage, called *charge dispersion*. Each of the plots represents a different E_J/E_C ratio. It can be seen that the charge dispersion ceases for a higher E_J/E_C ratio. Furthermore the anharmonicity decreases when increasing E_J/E_C.

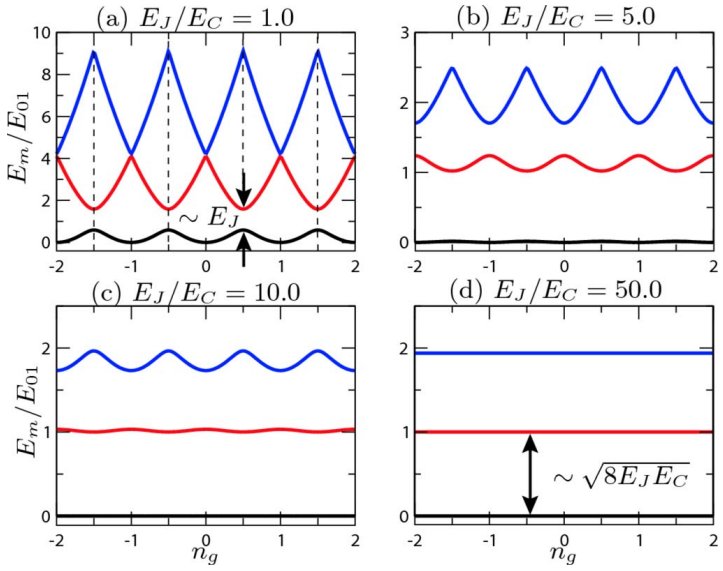


Figure 2.4: The lowest three eigenenergies E_m of the CPB, depending on the gate voltage, for 4 different settings of the E_J/E_C ratio. The energies are normalized by the lowest transition energy E_{01}, evaluated at $N_G = 1/2$. Figure adapted from [35].

Aiming for low charge dispersion, in 2007 Koch et. al. introduced the so-called Transmon qubit.

2.3 The Transmon Qubit

The main feature of the Transmon qubit is its high E_J/E_C ratio. This allows to have low charge sensitivity (see Figure 2.4) but still a sufficiently high anharmonicity. In particular, the Transmon

qubit manages to reduce charge dispersion exponentially in E_J/E_C while the anharmonicity only ceases algebraically [35]. Additionally, the Transmon design increases the coupling of a qubit to a resonator for higher E_J/E_C.

The main difference between the Transmon design and the one for the Cooper pair box is an additional shunting connection of the superconducting metal leads via a large capacitance C_B, as shown in Figure 2.5 (b). Additionally, the gate capacitance C_G is increased by a similar amount. The additional shunted capacitance is introduced by increasing the size of the two islands (in that case the reservoir becomes one of the islands). The additional capacitance lowers the charging energy $E_C = e^2/2C$ and therefore increases the E_J/E_C ratio. Figure 2.5 (a) shows a schematic drawing of a Transmon qubit that was used for most of the experiments during this project. The two big vertical blocks represent the two islands, coupled via the SQUID (denoted by JJ=Josephson Junctions; thin vertical line on the top of Figure 2.5 (a)) on the top. The Josephson Energy E_J can be tuned with the flux line (FL) on the top of the sample. The qubit can be driven by the charge drive line (DL) on the right side of the qubit. Additionally, there are two coupling resonators (CR1, CR2) that couple the qubit to its neighbors on the sample and there is a readout resonator (RR) used to readout the population of the qubit. The corresponding electrical circuit is shown in Figure 2.5 (b).

In general, having found a suitable qubit design the next question to ask is how to do experiments with it. To perform experiments in the framework of cavity QED the qubit needs to be coupled to a electromagnetic field.

2.4 Circuit QED

The well-understood framework of cavity quantum electrodynamics [29] can be extended to electric circuits, as shown in 2004 by Blais et. al. [8]. In this so-called circuit QED architecture the real atom is replaced by an artificial one (i.e. a superconducting qubit) and instead of a cavity a microwave transmission line resonator is used. The first implementation of such a setup was demonstrated by Wallraff et al. [49].
The advantage of circuit QED over cavity QED is that most parameters of a chip with superconducting qubits can be designed, such as the couplings between the resonators and the qubits or the qubit frequency range. Furthermore, several qubits can be placed on a sample of the size of 1 cm^2 which is beneficial for scalability [3]. At the same time circuit QED uses the same mathematical tools as cavity QED.
In particular, the interaction between a Transmon qubit and a resonator is described by the Jaynes-Cummings Hamiltonian [35]

$$\hat{H}/\hbar = \sum_{i=1}^{N-1} \omega_i \left|i\right\rangle \left\langle i\right| + \omega_r \hat{a}^\dagger \hat{a} + \sum_{i=0}^{N-2} \left(g_{i,i+1} \left|i\right\rangle \left\langle i+1\right| \hat{a}^\dagger + g_{i+1,i} \left|i+1\right\rangle \left\langle i\right| \hat{a}\right) \tag{2.11}$$

where the rotating wave approximation was used (as $\omega_{i,j} + \omega_r \gg |\omega_{i,j} - \omega_r|$). The first term describes the qubit with N energy levels and the second term characterizes the bare electromagnetic field in the resonator (with $E_0 = \hbar\omega_r$ set to 0). The third term then describes the interaction between the qubit and the resonator, including only transitions between neighboring energy levels.

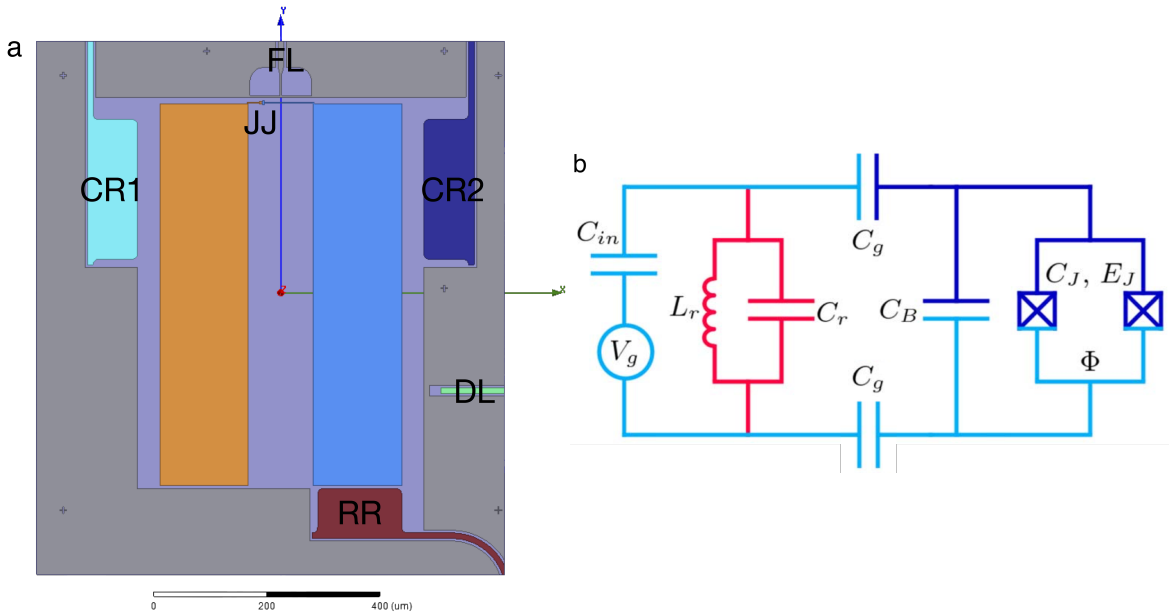


Figure 2.5: (a) Schematic, false-colored drawing of a Transmon qubit used in the Qudev laboratories. The big blocks in the middle are the two islands, connected by a SQUID, i.e. two Josephson Junctions (JJ). The line on the top represents the flux line (FL) that tunes the magnetic flux bias. On the top left and right one can see the coupling resonators to other qubits (CR1, CR2) that are capacitively coupled to the qubit. On the right side in the middle one finds the charge drive line (DL) for the qubit. On the bottom one end of the readout resonator can be seen (RR). Original image by Yves Salathé (Qudev).
(b) Circuit diagram of a Transmon qubit. From left to right one can see the gate voltage source, a coupled resonator (in red), the two gate capacitors C_G, the additional shunted capacitor C_B between the islands and the SQUID loop on the right (C_J,E_J). The flux line is not included to this circuit. Figure adapted from [35].

This approximation is valid for a sufficiently high anharmonicity of the qubit. The indices $g_{i,j}$ denote the coupling strengths between the microwave field and the qubit transition from state $|i\rangle$ to $|j\rangle$.
For high anharmonicity we can assume a two-level system that simplifies Eq. 2.11 to

$$\hat{H}/\hbar = \omega_{ge}\sigma_z + \omega_r \hat{a}^\dagger \hat{a} + g_{ge}(\hat{a}\sigma_+ + \hat{a}^\dagger \sigma_-) \tag{2.12}$$

with $\sigma^\pm = \sigma_x \pm i\sigma_y$. The eigenstates of this coupled qubit-resonator system are the so-called dressed states:

$$|+,n\rangle = \cos\Theta_n \,|e,n-1\rangle + \sin\Theta_n \,|g,n\rangle \tag{2.13}$$

$$|-,n\rangle = -\sin\Theta_n \,|e,n-1\rangle + \cos\Theta_n \,|g,n\rangle \tag{2.14}$$

and the ground state $|0,0\rangle$. Here, Θ_n is the mixing angle $\Theta_n = 1/2 \arctan(2g\sqrt{n}/\Delta_0)$ with the detuning $\Delta = |\omega_{ge} - \omega_r|$ and the resonator-qubit coupling strength g. The corresponding eigenenergies are

$$E_{\pm,n} = n\hbar\omega_r \pm \frac{\hbar}{2}\sqrt{4g_{ge}^2 n + \Delta_0^2} \tag{2.15}$$

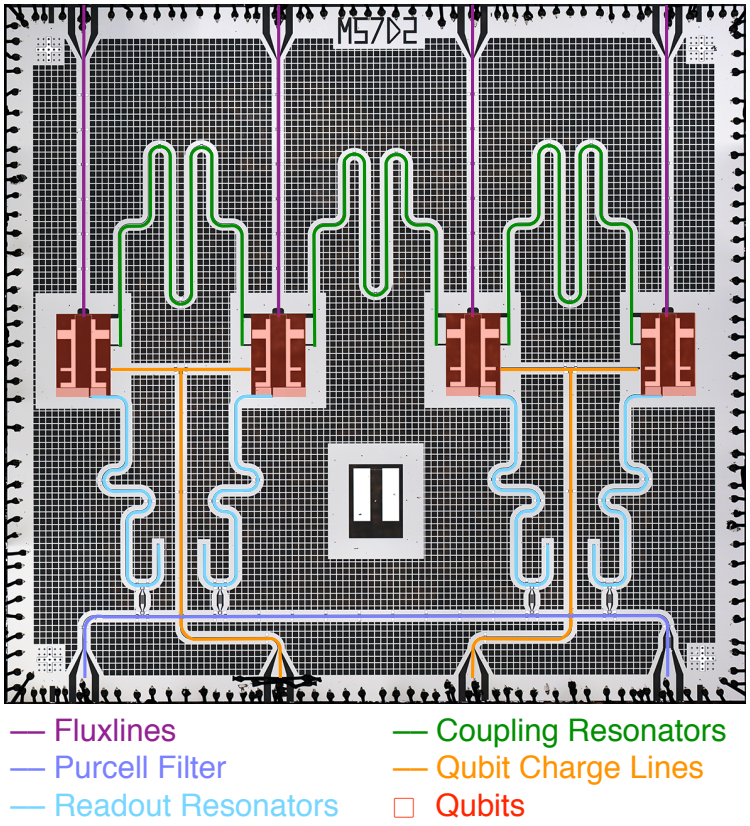


Figure 2.6: A (false-colored) picture of a 4 qubit chip used for quantum computing and simulation experiments in Qudev. Original photo by Yves Salathé (Qudev).

$$E_{0,0} = -\frac{\hbar\Delta_0}{2}. \tag{2.16}$$

The qubit can be operated at two different frequency regions: the resonant regime where $\omega_{ge} \approx \omega_r$ and the dispersive regime where the detuning Δ_0 is large. In the resonant regime the bare eigenstates $|g,n\rangle$ and $|e,n-1\rangle$ are not eigenstates of the total system anymore as the interaction term $H_I = \hbar g_{ge}(\hat{a}\sigma_+ + \hat{a}^\dagger\sigma_-)$ lifts the degeneracy. Instead, the two systems hybridize and the total eigenstates become $|\pm,n\rangle = \frac{1}{\sqrt{2}}(|g,n\rangle + |e,n-1\rangle)$ and $|0,0\rangle$. The two excited states differ in energy by a factor of $g_{ef}\sqrt{n+1}$ compared to the case without interaction (where the two energy levels are degenerate at $\Delta = 0$). This is called AC Stark shift [22]. Continuously, the qubit and the resonator exchange excitations which results in Rabi oscillations.

In the dispersive regime the two systems still influence each other, even though the qubit and the resonator do not exchange any excitations. In that case the interaction term of the Hamiltonian 2.12 can be neglected to lowest order. Instead we write [35]

$$\hat{H}/\hbar = \frac{1}{2}(\omega_{ge} + \chi_{ge})\hat{\sigma}_z + (\omega_r' + \chi\hat{\sigma}_z)\hat{a}^\dagger\hat{a} \tag{2.17}$$

where we used a renormalized resonator frequency $\omega_r' = \omega_r - \chi_{gf}/2$ as the resonator resonance gets slightly shifted due to the interaction with higher energy levels of the Transmon [3]. It can

be seen that the qubit state shifts the resonance of the resonator by a so-called dispersive shift

$$\chi = \chi_{ge} - \chi_{ef}/2 \tag{2.18}$$

where

$$\chi_{ij} = \frac{g_{ij}^2}{\omega_{ij} - \omega_r}. \tag{2.19}$$

Vice versa, the qubit transition frequency gets slightly shifted by a Lamb shift of χ_{ge} due to vacuum fluctuations in the resonator.

This system is especially interesting as the qubit state has a direct influence on the resonator frequency, which one can measure without disturbing the qubit population - a mechanism known as *quantum non-demolition readout*.

2.5 Drive and Readout of a Transmon

Dispersive Readout To perform a readout of a qubit state the corresponding readout resonator is driven with a weak microwave signal

$$\hat{H}_{drive} = \hbar\varepsilon_d(\hat{a}^\dagger e^{-i\omega_d t} + \hat{a}e^{i\omega_d t}) \tag{2.20}$$

to populate it with photons. ε_d denotes the amplitude of the signal [8]. The sample design depicted in Figure 2.6 contains a Purcell filter that couples to the readout resonators to suppress Purcell decay [46]. In that case all readout resonators are driven through the same input port and their response is read out through the same output port (namely the input/output port of the Purcell resonator).

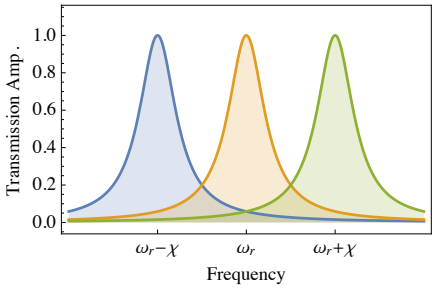


Figure 2.7: Transmitted amplitude of a microwave signal passing through a readout resonator when doing qubit spectroscopy. Orange: Unshifted resonator. Blue (Green): Response due to coupling qubit in the ground (excited) state. Figure adapted from [24].

The resonator is typically far detuned from the qubit transition frequency such that we can apply the calculations for the dispersive regime from Section 2.4. As seen in Eq. 2.17 and Figure 2.7 the resonator shift depends on the state of the qubit. For a qubit in the excited state the resonator frequency shifts $\omega_r \to \omega_r + \chi$. This shift can be detected by measuring the response of the driven resonator.

Single qubit gates The qubit can be driven through its charge line which is shown in Figures 2.6 and 2.5 (a). To see the effect of a driving microwave signal applied through the charge line an additional term can be added to the Hamiltonian in Eq. 2.17. This additional term represents an oscillating microwave field with frequency ω_D: [3]

$$\hat{H}_{drive}/\hbar = \frac{\Omega}{2}\hat{\sigma}_z \tag{2.21}$$

with $\Omega = 4E_C C_g V_D/e\hbar$. Diagonalizing the Hamiltonian leads to

$$\hat{H}_{drive}/\hbar = \Omega_R \cos(\omega_D t)\hat{\sigma}_x \tag{2.22}$$

with Rabi frequency $\Omega_R = \Omega \sin(\theta)/2$ where $\theta = \arctan(\frac{E_J}{4E_C(1-2N_G)})$. In total we now have

$$\hat{H} \approx \frac{1}{2}\omega_{ge}\hat{\sigma}_z + \Omega_R cos(\omega_D t)\hat{\sigma}_x \tag{2.23}$$

which shows that using flux pulses (change ω_{ge}) and microwave drive pulses (correspond to $\Omega_R(\omega_D t)$) one can perform arbitrary qubit operations on the Bloch sphere.

Qubit-Qubit interaction Finally, to perform two-qubit gates the qubits on a chip are connected by a coupling resonator as seen in Figure 2.6. The Hamiltonian 2.2 can then be extended by an additional term that describes the interaction of a qubit with one of its neighbors [7]

$$H_{I,2q} = \hbar \frac{g_1 g_2(\Delta_1 + \Delta_2)}{2\Delta_1\Delta_2}(\sigma_1^+\sigma_2^- + \sigma_1^-\sigma_2^+) \tag{2.24}$$

where higher order terms in g_i/Δ_i were neglected. The $\Delta_i = \omega_{q_i} - \omega_{r_i}$ terms denote the detuning of the i-th qubit from its resonator readout frequency.

2.6 Coherence

A quantum state that couples to the environment degenerates, i.e. it decays to the thermal equilibrium state of the system and the phase information gets lost. Long coherence times for qubits are crucial for quantum information processing and in particular for quantum computing.
One distinguishes two classes of loss of quantum information [17]. First, the Bloch vector decays to the ground state. This type of error is called *energy relaxation*. In the Bloch sphere picture, this process is represented by a Bloch vector that evolves towards the ground state. Second, the Bloch vector can also diffuse around the z-axis of the Bloch sphere. In that case the information about the phase of the state is lost, which is why this process is called *dephasing*.

To formulate a mathematical expression of the coherence times it is helpful to consider the Bloch sphere picture with eigenvectors $\hat{x}$, $\hat{y}$ and $\hat{z}$. A quantum state with Bloch vector $\vec{S}_0$ at $t = 0$ will decohere and reach its equilibrium state represented by $\vec{S}^{eq}$ with $S_z^{eq} = \tanh(\hbar\omega_{ge}/2k_BT)$.
The relaxation and decoherence rates are now defined as [17]

$$\Gamma_1 = \lim_{t\to\infty} \frac{\ln\langle S_z(t) - S_z^{eq}\rangle}{t} \tag{2.25}$$

$$\Gamma_\varphi = \lim_{t\to\infty} \frac{1}{t} \ln\left(\frac{\langle \vec{S}(t)\vec{S}_0(t)\rangle}{|\vec{S}(t) - S_z^{eq}z|}\right) \tag{2.26}$$

where an exponential decay of the components of the Bloch vector $\vec{S}_0$ was assumed. The qubit relaxation time is then defined as

$$T_1 = \frac{1}{\Gamma} \tag{2.27}$$

where the coherence time is defined as

$$T_2 = \frac{1}{\Gamma_\varphi + \Gamma_1/2}. \tag{2.28}$$

The interpretation of the two distinct coherence times is as follows [17]: The T1 time describes the rate by that the qubit loses energy and decays to the ground state (*energy relaxation*). As it includes both contribution of energy relaxation and dephasing, the T2 time characterizes the total loss of quantum information of the qubit.

Chapter 3

Experiment setup

The experiments performed during this project were done in the "Quantum Device Laboratory" (Qudev) at ETH Zürich. This chapter provides an overview of the technical aspects of the experiments. In the first section the general experiment setup will be explained. Then it will be shown how a dilution refrigerator works. Afterwards the electrical parts of the setup will be described and finally it will be explained how the different steps of an experiment are synchronized.

3.1 Overview

There are three main components of the setup for the quantum computing experiments at the ETH Qudev laboratory: the software to control the experiments, the electrical parts to send microwave signals to the chip and to read out the response and cryogenics.
A typical experiment follows a sequence of steps which are depicted in Figure 3.1: First, the experimenter starts the measurements using a software written in LabView programming language. At that stage the experiment is defined and the parameters for the measurement are set. Using this information the software then sets the settings for the instruments that are used for the experiment. Typically, these are Arbitrary Waveform Generators (AWGs), Microwave Generators (MWGs) and an acquisition device. The quantum computing group uses a Field Programmable Gate Array (FPGA) to analyse the data from the chip. The software ensures that the synchronization is correct, i.e. it decides which instrument parameters have to be changed at what stage of the experiment.
The microwave signals are produced by the microwave generators. These signals are then modulated (see Section 3.3) and sent through cables to the chip. The chip is placed at the bottom of the cryostat. On the chip, the different signals interact.
Afterwards the readout signal leaves the cryostat, gets amplified and downconverted before it is recorded by a FPGA. The FPGA sends the data back to the LabView software where the results are presented.

3.2 Cryogenics

To ensure the chip is superconducting and to reduce thermal noise (see Section 2.1) the chip is placed inside a dilution refrigerator from Oxford Cryogenics. A scheme of the "Kelvinox" model

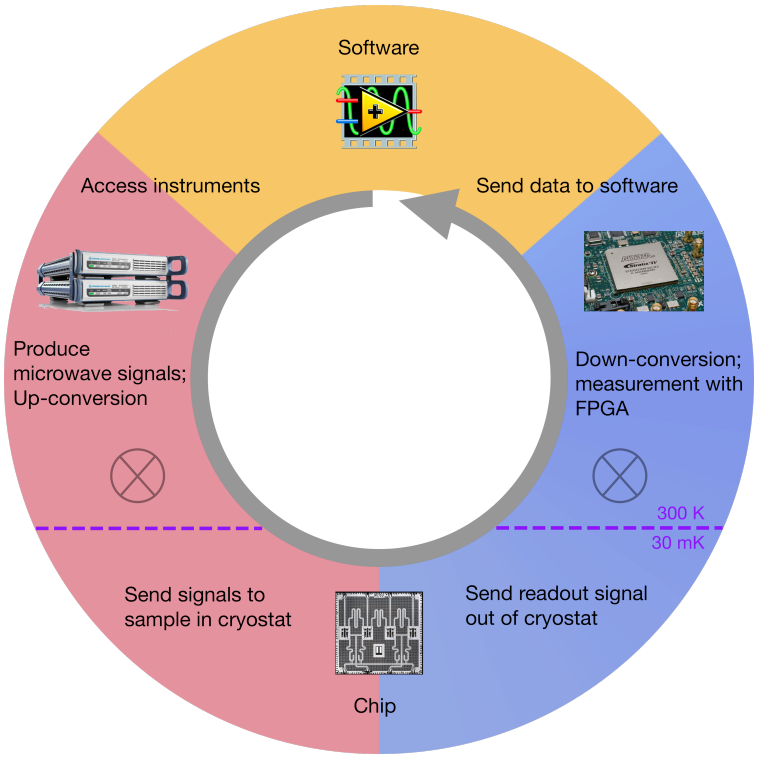


Figure 3.1: The typical steps of a quantum computing experiment at the Qudev laboratory with the three main components of the setup (software, yellow; up-conversion, red; down-conversion and readout, blue). The experiment is controlled by a LabView application. First, the signals are generated, upconverted and sent to the chip in the cryostat. The readout signal is then down-converted and read out by an FPGA from which the software reads the data. Figure includes material from [31], [42] and [15].

which was used is shown in Figure 3.2.

The cryostat works as follows [2]:

1. Gaseous ^{3}He flows from a dump outside of the cryostat to the condenser inside the 1K pot. These parts are placed on the top inside the refrigerator. As the condenser is much colder than the incoming gas temperature, the gas condenses.
2. The helium is then sucked into the mixing chamber on the bottom of the refrigerator, next to the experiment chip. Due to heat exchangers the liquid is cooled further.
3. In the mixing chamber there are two phases of a mixture of ^{3}He and ^{4}He: a concentrated phase rich in ^{3}He and a dilute phase rich in ^{4}He. As the enthalpy of the two phases is different it is possible to move liquid ^{3}He from the concentrated phase to the dilute phase. This quantum mechanical process can be simplified as a phase transition (similar to a "evaporation" of ^{3}He, even though we only deal with liquid helium) that removes energy from the environment and therefore results in a cooling.
4. For a continuous phase transition one has to keep on disturbing the equilibrium. First, ^{3}He has to be continuously added to the concentrated phase. At the same time, ^{3}He has to be removed from the dilute phase to prevent it from saturation. This is what the still is for.

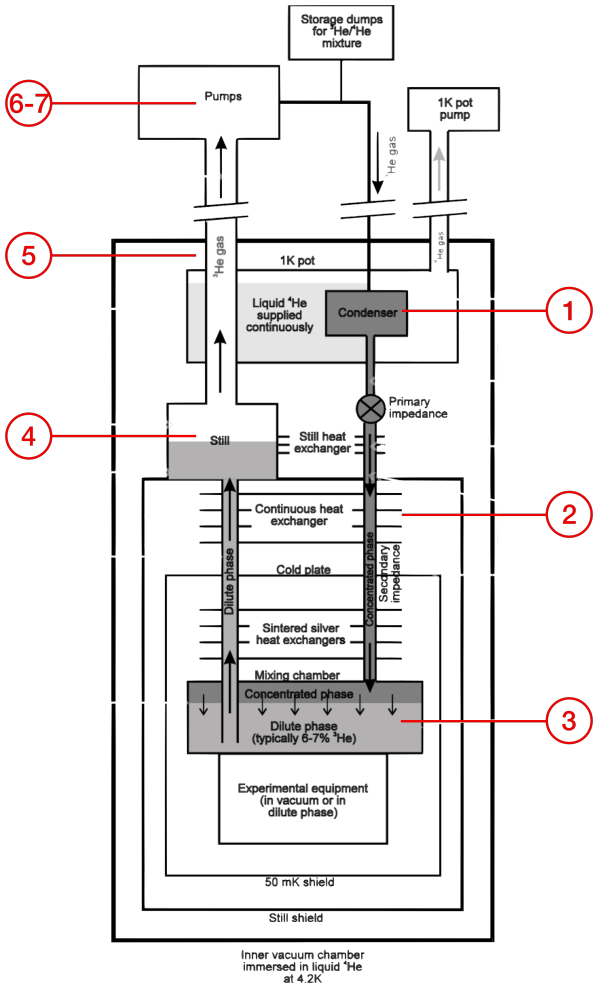


Figure 3.2: Schematic drawing of the Oxford Kelvinox dilution refrigerator. The red numbers represent the steps of the list in Section 3.2. Figure adapted from [2].

The still is a small container on top of the mixing chamber that warms up the gas in the dilute phase. Thanks to the different evaporation temperatures of the two isotopes it is mostly ^{3}He that evaporates.

5. This gas is now pumped out of the cryostat and warmed up to room temperature. As a side effect this gas cools down the incoming helium at the opposite side of the fridge through heat exchangers.
6. Outside of the fridge the ^{3}He is sent to the "cleaning" cycle. There the gas is pumped through a cold trap filled with liquid nitrogen. Like this the gas is cleaned from oil that may have entered the gas when flowing through the pumps.
7. As the ^{3}He got sucked out of the fridge its pressure in the still decreases and therefore new liquid flows from the mixing chamber to the still. At the same time the ^{3}He that went through the cleaning cycle is pumped to the fridge again where it enters the condenser. The cycle repeats.

Theoretically using that technique the mixing chamber can be cooled to absolute zero tem-

perature. However practically, there is heat load from the environment that limits the minimal temperature in the mixing chamber. On one hand heat enters the cryostat through the cables. This can be suppressed to a certain limit through a smart use of attenuators. On the other hand heat is supplied form the environment in the laboratory. To suppress this effect the mixing chamber is shielded by an outer vacuum chamber, a container full of liquid helium (4.1K), a inner vacuum chamber and a so-called still shield to shield the chip from radiation heat load. Using this technique the temperature of the mixing chamber of the Oxford Kelvinox refrigerator in the Qudev laboratory typically reaches 28mK.

3.3 Signal modulation

In this section it will be explained how microwave signals are produced and sent to the chip. Each MWG that was enabled by the control software produces microwaves. In general, one is not

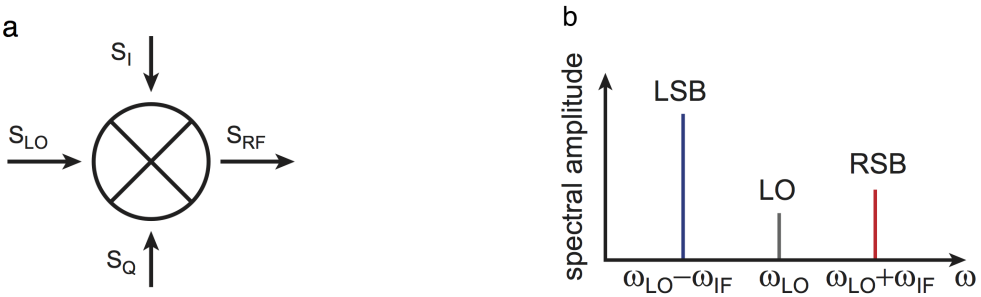


Figure 3.3: (a) Schematic representation of quadrature upconversion using a mixer. (b) The output signals after upconversion. In general two sidebands as well as the LO signal can be seen. Figures reproduced from [3], pages 67-68.

interested in the bare signal a MWG produces but in a modulation of this signal with another one, typically produced by an AWG. For that a corresponding pulse sequence pattern has to be loaded to the AWGs. Before the microwave signals are sent to the cryostat they are in general modulated with other signals using quadrature upconversion. There the bare signal that is produced by the microwave generator

$$s_{LO} = Acos(\omega_{LO} t) \tag{3.1}$$

is sent into the local oscillator port (LO) of the mixer (see Figure 3.3 (a)). A is the amplitude of the signal, ω_{LO} its frequency. This signal is then split into its I and Q components [3]. The I part is multiplied with another signal coming from an AWG. That signal oscillates with frequency ω_{IF} and is in phase with the carrier

$$s_I = I \cos(\omega_{IF} t + \varphi) \tag{3.2}$$

whereas the Q part of the local oscillator is multiplied with a signal phase shifted by φ_Q,

$$s_Q = Q \cos(\omega_{IF} t + \varphi_Q + \varphi). \tag{3.3}$$

The combined RF signal that goes to the fridge then consists of two sidebands signals. Their frequencies are shifted by the IF frequency such that $\omega_{RF} = \omega_{LO} \pm \omega_{IF}$. One of the sidebands

can be canceled by setting the phase of the mixing Q component to $\varphi_Q = \pm\pi/2$ (called *single sideband mixing*). Then a single signal remains that oscillates at a frequency that is shifted by ω_{IF} from ω_{LO}. The amplitude of the signal can be tuned using the I and Q amplitudes generated by the AWG and the phase can be adjusted by φ. However, due to mixer imperfections the other sideband as well as the carrier signal do not get fully suppressed as shown in Figure 3.3 (b). To decrease the unwanted signals further one can perform a mixer calibration and use filters.

3.4 Downconversion

On the chips used for this Master's thesis a Purcell filter couples to all readout resonators (see Section 6.1). A Purcell filter takes the role of an additional resonator with a resonance that is broad compared to the ones of the readout resonators. As the Purcell filter couples to each of the readout resonators any of them can be accessed by sending a pulse to the input port of the Purcell resonator. Likewise, there is only one output port (i.e. the output port of the Purcell resonator) that carries the response signals of the readout resonators. A signal coming from the sample is first amplified by a parametric amplifier. This is necessary as the resonator on the chip is only populated by a few photons. After the amplification the signal is downconverted to typically 25 MHz for a heterodyne detection. For that it has to be ensured that there is a microwave generator that produces a downconversion LO signal that is always 25 MHz below the readout frequency. After a further amplification [43] the signal is split into its I and Q parts which are then passed to the FPGA. The FPGA records the data and provides the data to LabView when asked for it.

3.5 Synchronization

Since there are various devices and signals that are involved in a single measurement, synchronization between the different steps and devices is essential to do sensible experiments. This section provides an overview of how the different devices and signals are synchronized.
The main AWG controls the timing. The patterns loaded to the AWG determine when the main AWG should trigger the other devices. An example for a pattern for pulsed spectroscopy is shown in Figure 3.4:

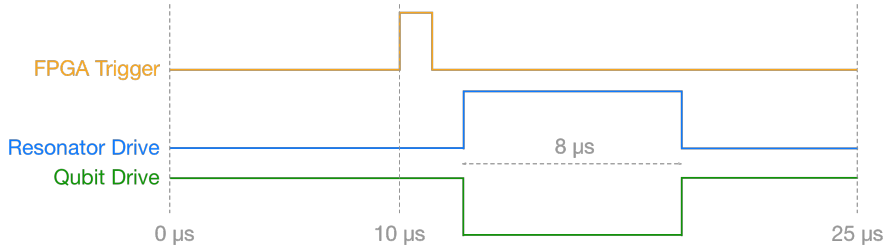


Figure 3.4: A AWG sequence pattern that is used for pulsed spectroscopy. The yellow line represents the trigger for the FPGA. The blue line shows the pattern for the resonator drive and the green line represents the qubit drive signal.

1. First, the AWG triggers the FPGA to start with the measurement (yellow trigger in Figure 3.4).

2. After a short delay the AWG triggers the MWG that is responsible for the resonator drive to start producing microwaves. At the same time the MWG that produces the microwave for the qubit drive is stopped. 8μs later this is reversed again. More about pulsed spectroscopy can be found in Section 6.3.
3. A trigger device specifies how often this pattern is repeated. It periodically sends a pulse to the AWG, indicating that the arbitrary wave generator has to restart the pattern.

There are several parameters that define how long an FPGA measurement takes. The FPGA acquires data with a rate of 100 million samples per second (100MHz). Therefore a single FPGA sample takes 10ns. Using the FPGA decimation factor parameter (abbr. "dFactor") one can choose how many of these measurements should be skipped. A "dFactor" value of 2 e.g. means that only every 2nd sample of the FPGA is stored in the dataset. The number of samples specifies the number of measurement sets the FPGA should take during one sequence. Then, the number of segments tells the FPGA how many different patterns are loaded to an AWG. In the above example, there is only one. Moreover, the number of averages specifies how often such a pattern should be repeated and averaged over.

As an example it can be considered a dFactor of 2; number of sequences = 1; number of averages = 1000. This means that there is only one pattern loaded (e.g. the one in Figure 3.4) and that this pattern is repeated 1000 times such that the FPGA can later average over all of the 1000 datasets. More averages increases the signal to noise ratio as the noise is averaged out more radically. The remaining question is how to choose the number of samples wisely. Let us assume the main trigger is fired every 25μs. This means there are 25μs between two FPGA sequence triggers. Considering that every sample takes 10ns and that only every second (dFactor = 2) one is taken there can be recorded at most $\frac{25\mu s}{2*10ns} = 1250$ samples during these 25μs. However, depending on the used AWG sequence pattern one may need to lower the number of samples to not measure a lot of noise after the qubit has decayed. This can be illustrated with the example sequence pattern shown in Figure 3.4. Here, the part of the pattern with AWG pulses begins at 10μs, where the FPGA receives the "start recording" trigger. The number of samples the FPGA must record therefore has to be calculated starting from this point in time. In principle, data can now be recorded until the next segment begins. However, as the qubit lifetime may be shorter than the remaining time of the pattern, it is wisely to adjust the recording time, i.e. the number of samples, to not include a lot of noise data at times where the qubit has already decayed. Moreover, the main repetition rate, which can be set individually from the FPGA and AWG settings, should be set such that it does not trigger the AWGs before their patterns have finished. Also it should be prevented that the repetition rate is too high such that when the new pattern begins the qubit is still excited due to pulses from the former pattern. Finally one can manually adjust the delay time between the FPGA trigger and the start of the MWGs. For spectroscopy experiments where the total time trace is integrated out, the data in between the two triggers is noise that lowers the signal to noise ratio.

The whole measurement cycle is shown in Figure 3.5. First, the software sets the parameters of the instruments for the current measurement. These are e.g. new frequencies or power parameters for a microwave generator. Then *SweepSpot* calls the FPGA to ensure the device is ready to gather data. Next, the main trigger sends a notification to the main AWG to start the pattern that is loaded. The AWG then triggers the relevant devices at the times defined by the pattern. This includes microwave generators, other AWGs and of course the FPGA. As soon as the FPGA

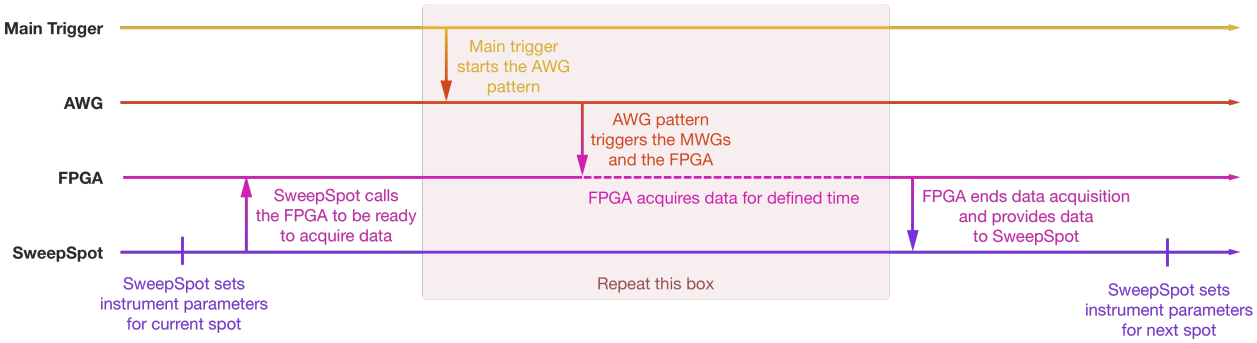


Figure 3.5: The timing sequence of events during one FPGA measurement cycle with SweepSpot. Each line corresponds to one component of the setup.

receives the ”start recording” trigger it begins with the data acquisition. The acquisition ends as soon as the pre-defined number of samples has been reached (see above paragraphs). In general these three steps, as shown in Figure 3.5, are repeated for as many times as defined by the number of segments that have been specified in the FPGA settings. Furthermore, all segments are repeated for the predefined number of FPGA averages. For each FPGA sample, the FPGA performs post-processing algorithms on the data such as averaging. In LabView the new dataset is appended to the former data and the next spot of the experiment begins, i.e. the cycle shown in Figure 3.5 is repeated with other settings.

The part of the setup which is most relevant for this project is the software component. Now that the basic procedure of an experiment is clear the software part can be introduced. In the following two chapters the software is explained in much greater detail. It will be shown how the software controls the experiment, how it is used and what has been improved thanks to this project.

Chapter 4

Software

A complete rewriting of the software used to perform experiments with in the Qudev laboratories is one of the core tasks of this Master's thesis. In this chapter this new software suit, called "SweepSpot", is presented. It will be shown how SweepSpot works, what its advantages are and how it is used in the laboratory.

SweepSpot is a software suit developed in the Qudev laboratory using LabView 2014 as a part of this Master's thesis. SweepSpot on the one hand replaces the old measurement software called "Cleansweep". Because of its different concept and cleaner implementation it is more powerful and also more flexible for implementations of further additions. Additionally it comes with new features and simplifications for the experimenter. On the other hand, thanks to its new concept SweepSpot can be used as a module itself, embedded in some higher-level VI. This opens up a range of new possibilities. This will be the subject of Chapter 5. In this chapter SweepSpot is described in detail. It will be shown how SweepSpot operates, how one can use its new features and what the improvements compared to Cleansweep are.

4.1 SweepSpot framework

4.1.1 Concept

The heart of the SweepSpot software environment is the LabView VI *SweepSpot main*. It can be called by any higher-level function to perform measurements using the SweepSpot framework. A natural example is *SweepSpot frontend* which is the basic frontend VI to perform experiments with SweepSpot. There, *SweepSpot main* is the main module that is executed in the background. The same applies if experiments are performed using the LabView software *QubitCalib* which is a higher-level VI used to perform sequences of experiments (see Section 5.1).
SweepSpot main consists of 3 submodules as illustrated in Figure 4.1: the first one, *SweepSpot generator*, collects the defining settings about the sweep and recombines them to a spot list, as described in Section 4.1.3. The actual measurement is then done by *SweepSpot.vi* and in the third part, *Save measurement data*, the software writes the corresponding data files to the database. The next paragraphs describe the individual steps in *SweepSpot main* in more detail.

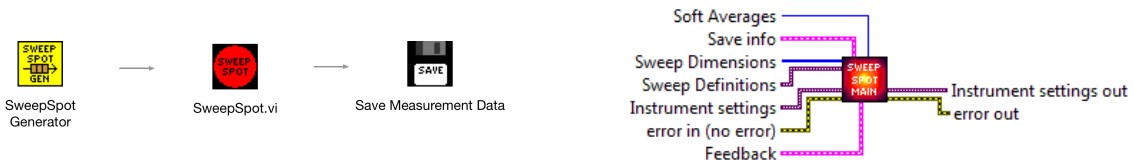


Figure 4.1: (a) Schematic representation of *SweepSpot main.vi*. First the experiment is prepared (*SweepSpot generator.vi*), then the experiment is performed (*SweepSpot.vi*) and afterwards the data is saved (*Save measurement data.vi*). (b) The input and output data of *SweepSpot main.vi*. The parameters on the left are input values and the ones on the right are output parameters.

4.1.2 Definition of an experiment

This section describes the first step of any measurement, namely how SweepSpot defines the type of the experiment and its settings. There are 4 main datasets that carry certain information about the experiment, as shown in Figure 4.1: *Sweep Dimensions*, *Sweep Definitions*, *Instrument settings* and *Save info*. All are created in a higher-level VI (e.g. *SweepSpot frontend* or *QubitCalib*) and are passed to *SweepSpot main*.

Sweep dimensions and Sweep definitions

A sequence of measurements of the transmission signal through a sample that is performed at different instrument settings is called a *sweep*. Typically only few parameters are changed throughout the experiment. Each step in this sequence is called a *spot*. *Sweep dimensions* is an array of LabView ”ring” elements (basically just strings) that contain the information about what parameters are swept. The first element of the array corresponds to the fastest dimension which is ”Frequency 1” in the example shown in Figure 4.2. The second element corresponds to the next slower dimension. One can for example think of additionally sweeping the qubit drive power. In that case a frequency sweep (dimension 1) is done for each power strength (dimension 2). Cleansweep was limited to 2 sweep dimensions. SweepSpot can handle an arbitrary number of dimensions. For example in Figure 4.2 a third dimension ”DC Voltage” can be imagined that would perform a magnetic field sweep (dimension 3) for each of the MWG power and frequency settings.

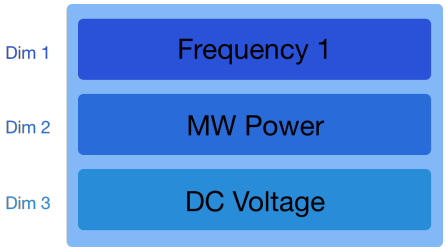


Figure 4.2: The *Sweep Dimensions* array for 3 dimensions.

Sweep definitions is a class of objects that contain the defining information about each sweep dimension. Currently there are 6 supported Sweep Types. For each one there is a specific data type defined in LabView.

- *Frequency Sweep* changes the frequency of a given microwave generator through a defined range. It is defined by the start frequency, the stop frequency, the stepsize and the index of the MWG. This is the only sweep type that has two instances such that one can sweep combinations of frequencies of two different MWGs.
- *MW Power* sweeps the output power of a microwave generator through a given range. A power sweep is defined by the start power, the stop power, the stepsize and the index of

the MWG.

- *DC Voltage* can be used for sweeping the voltage on a magnetic field coil. It changes the voltage on one of the coils or flux lines through a defined range. This sweep is defined by the start voltage, the stop voltage, the stepsize and the index of the DC source.
- *AWG sequences* sweeps through a list of AWG waveform sequence description files that are loaded to the different arbitrary wave generators of the setup. This sweep is defined by a list of paths to the desired sequencefiles.
- *Laser position* is used in experiments where moveable piezoelectric displacement actuators are installed (such as in the Qudev "Arctic" laboratory). With this sweep type one can automatically move the displacement element. The corresponding sweep is defined by the origin X and Y position of the actuators, the pixel size, the number of points to sweep over and the path type.
- *Repeat* can be used to repeat all the faster dimensions by the specified number of times.

Sweep Definitions is structured as a LabView variant which is similar to a dictionary, as illustrated in Figure 4.3. As a dictionary, it contains an element for each sweep type. One can easily access any element using the "Get Variant Attribute" LabView function to get the corresponding entry of the variant. These objects (still of type "variant") then need to be type casted to elements of the corresponding type definitions using another LabView function. These objects can then be used to e.g. read out parameters of a sweep.

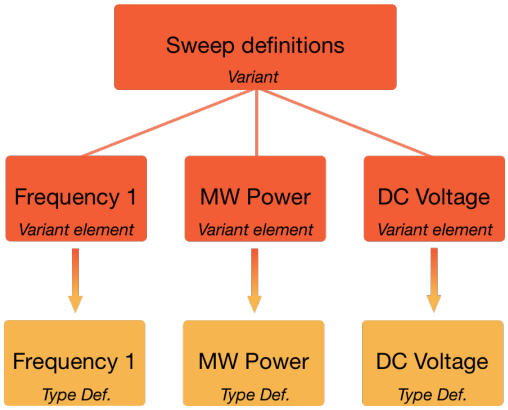


Figure 4.3: The *Sweep Definitions* variant, represented as a big red box. Its elements, still LabView variants, are represented as small red boxes. The type definitions they can be converted to are shown as yellow boxes.

Instrument settings

Instrument settings is an object that carries parameters of the instruments that are set before the sweeps. Similarly to *Sweep Definitions*, it is a LabView "variant" with an element for each instrument of the setup. Such elements can be FPGA parameters, MWG and AWG settings, Trigger parameters, information about DC sources, controls for switches and settings for the laser microscope. The parameters defined in *Instrument settings* are set once in the beginning of the experiment and are in general not changed. The only exception is if a sweep type later overwrites certain values. It may be helpful to look at the above example again. There the fastest dimension was a frequency sweep and the second a DC voltage sweep. In the beginning of the experiment SweepSpot sets the values of all microwave generators and DC sources which are defined in *Instrument settings*. Then during the experiment, those (and only those) values defined by *Sweep Definitions* are changed in each sweep. In our example these are the microwave frequency of a particular MWG and the voltage on one of the coils. Other parameters such as the GPIB/Visa address of the devices or the MWG phase adjustments are not modified as these are not part of the Sweep Definition of a frequency sweep or a DC voltage sweep. In the following list the individual elements of *Instrument settings* are described. Each element is a LabView type definition.

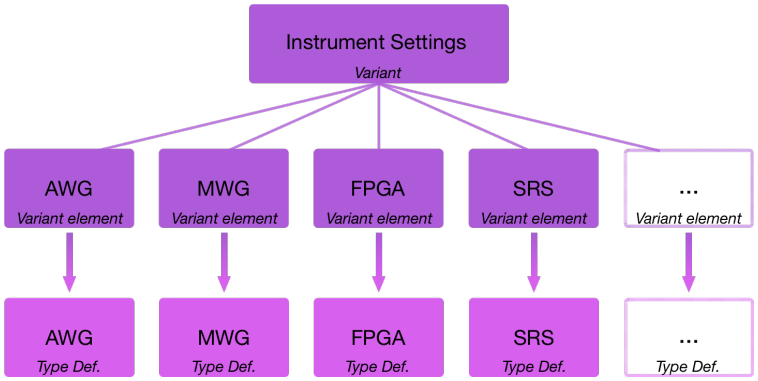


Figure 4.4: The *Instrument Settings* variant, its elements and the conversion to type definitions.

- **AWG** The AWG control contains information about the Arbitrary Wave Generators that are connected to the setup, as shown in Figure 4.5.

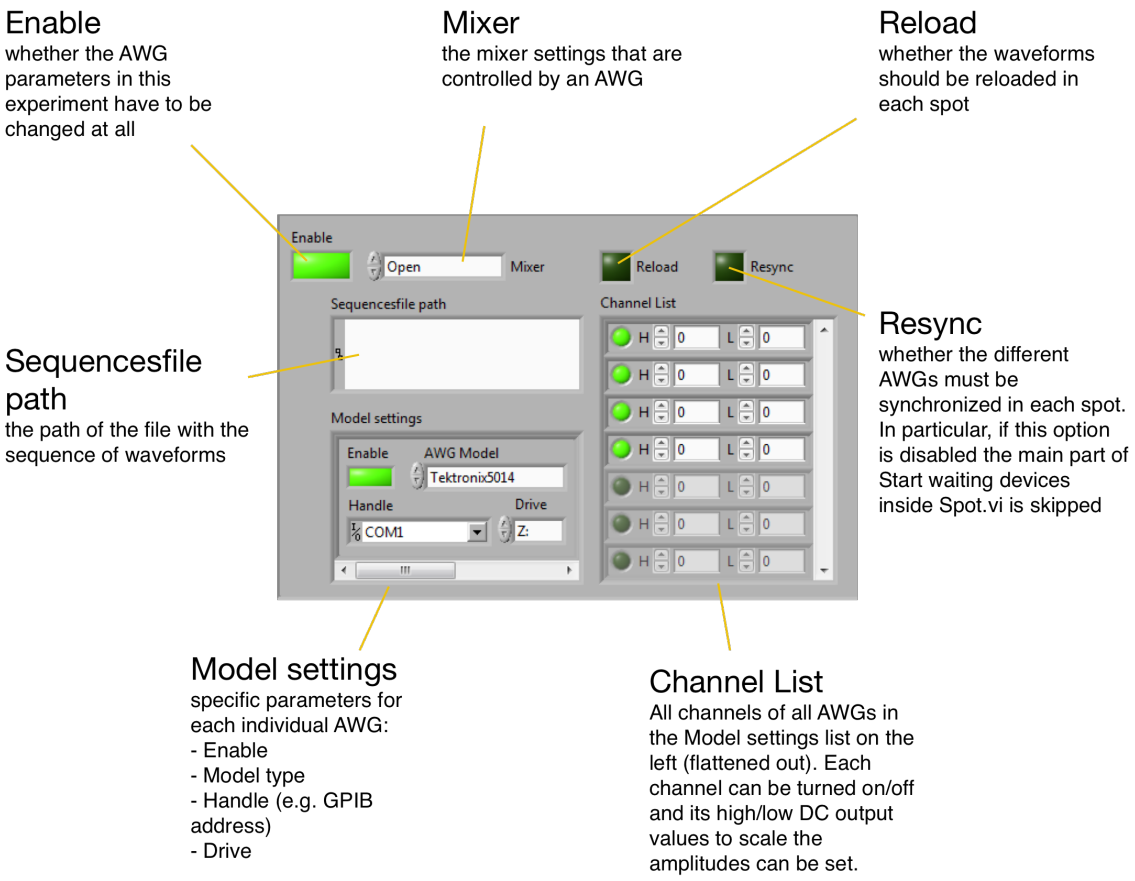
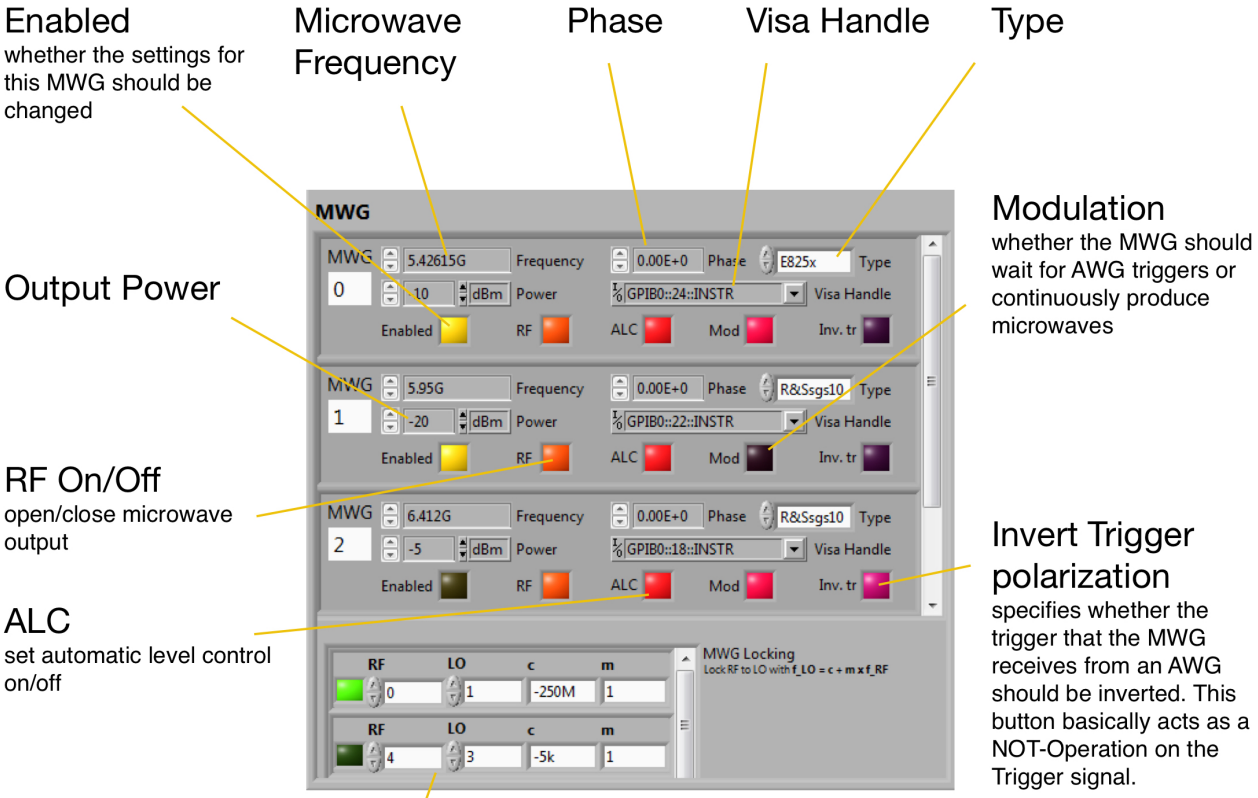


Figure 4.5: The AWG instrument control.

- **MWG** The MWG section contains information about the microwave generators that are used in the setup, as shown in Figure 4.6. In the MWG list each element represents one generator.



$$\nu_{LO} = c + m * \nu_{RF}$$

Figure 4.6: The MWG instrument control. The controls on the upper list set parameters for each microwave generator individually. On the bottom there is a list that specifies which MWG frequencies should be locked to others.

- **FPGA** The FPGA section is an extensive type definition with parameters that specify how the FPGA performs the readout. As shown in Figure 4.7, the FPGA section consists of two sub-type definitions: *FPGA Parameters* and *FPGA Measurement Settings* including a Boolean *Average over time* that tells whether one wants to integrate the signal over time.

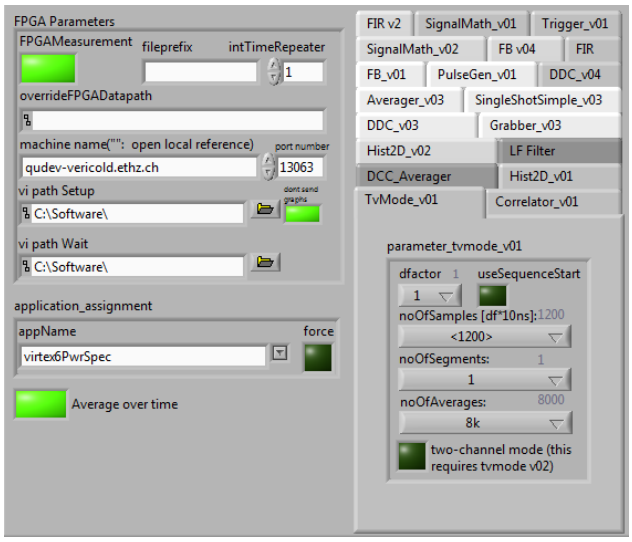


Figure 4.7: The FPGA instrument control.

FPGA Measurement Settings contains several sub/typedefs which specify various parameters that affect the FPGA readout. *FPGA parameters* contains more global parameters such as the FPGA address or the FPGA application (e.g. "tvmodeV02").

- **Trigger** The trigger section contains the address to the trigger source, the repetition rate and an enable button that indicates whether a new repetition rate should be set.
- **Stanford Research Systems DC Source (SRS)** The section for the SRS instrument (Figure 4.8) contains an enable button and the path to the DC source instrument.

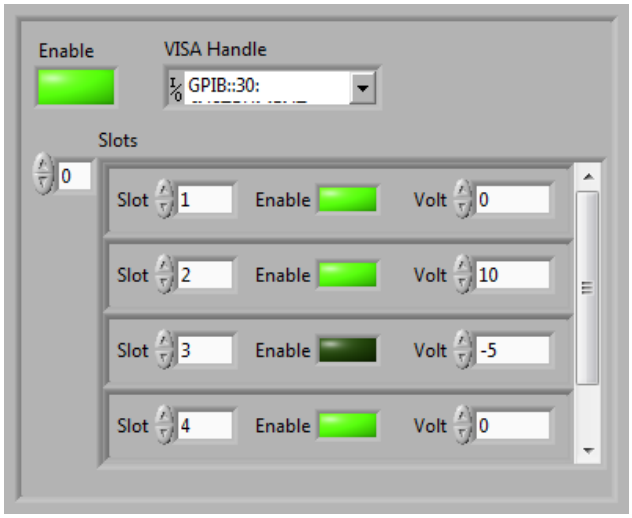


Figure 4.8: The Stanford Research Systems (SRS) DC Source instrument control.

Then it contains an array of the actual DC sources that set the voltages of a particular coil or flux line. Each element contains an enable button, a slot index and the voltage to be set. These elements also contain a hidden "address" element (e.g. a GPIB-address) which is automatically overwritten by the main path in the backend and is only needed for operational reasons. It is therefore at the moment not intended that the user changes

the individual paths of the DC sources on the frontend as long as there is only one DC source. This is the reason of the element being hidden at the moment. If there will be added multiple DC sources at some point one may delete the main address element and show the individual addresses on the frontend instead.

- **Laser microscope** This instrument type contains the path to the server that controls the piezoelectric elements as well as the server host. Additionally there are two parameters ”Z Out” and ”Delay time” that can be used to change the general behavior of the actuator.
- **RF Line Switch** and **SwitchBoard** These controls handle the status of switches. The *Switch-Board* control accesses the Advantech switch whereas the *RF Line Switch* section controls classic switches.

Instrument settings is a variant that contains elements for each device type that can be accessed by calling the device type name (e.g. ”AWG”) with the LabView function ”Get Variant Attribute”. *Instrument settings* is also one of the outputs of *SweepSpot main*.

Save info

Save info is a cluster that contains all relevant information about the saving of data- and configuration files with SweepSpot.

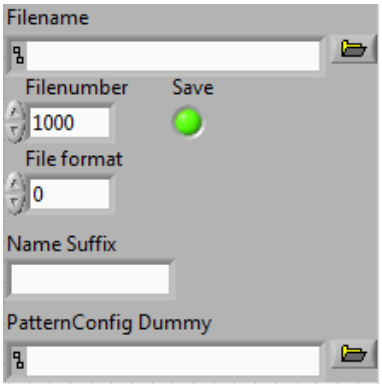


Figure 4.9: The *Save Info* type definition

As Figure 4.9 shows this includes:

- the name and path of the files that SweepSpot saves;
- the filenumber of the current experiment;
- the path to the pattern configuration file. This can be used by a higher-level VI if one wants to manually save a pulse configuration file for a Mathematica analysis of the data at a later stage (see Section 4.1.5);
- a boolean indicating whether SweepSpot should save data files or not;
- the file type, a setting that decides whether SweepSpot should save the data in ASCII or binary format;
- a filename suffix which can be used by a higher level VI to append certain information about the experiment. This can be useful e.g. if there are several experiments with the same base filename but performed on different qubits.

Soft (Software) averages

This is an integer that sets the number of times the complete experiment is repeated and averaged over. This type of averaging accounts for fluctuations in the data on a longer timescale. Software averages are also needed because the number of averages an FPGA can perform is limited. This is due to overflow problems in certain calculations on the FPGA where the result is represented by more bits than allowed by the fixed-point number. This is especially an issue as the FPGA uses fixed-point values. This is a problem for averaging as an overflow can happen for addition.

Feedback and Error

Additionally there are 2 more inputs to *SweepSpot main*. They do not actually define something about the experiment but for the sake of completeness they should still be mentioned here. The first one, *Feedback*, is a cluster of references to frontend elements. At the moment these are references to Sweep info (see Section 4.1.3), Measurement data (see Section 4.1.4), Live status cluster (a cluster that contains information about the current spot) and a frontend status box and progress bar. It is optional to wire the *Feedback* input. It can be used if one wants to gather information about the current state of the experiment and show it on a frontend. More information can be found in Section B.5. Then, there is a standard LabView error input that collects error messages throughout the experiment. *SweepSpot main* has two outputs: First, the modified *Instrument settings* and second the error output. Like this *SweepSpot main* can be used as a module in some higher-level VI (see Appendix B.5). In the next sections it will be described how these settings will be used to actually perform an experiment.

4.1.3 SweepSpot generator

SweepSpot Generator converts the defining settings of the experiment to a so-called *Spot array*. This is a 2D array that contains the parameters for the relevant devices that have to be set in each spot (see Figure 4.10). The first dimension of the *Spot array* runs over all the spots. Then for each spot there are as many entries as there are new instrument parameters to be set (this is the second dimension of *Spot array*). Later SweepSpot will loop over the *Spot array*, first over its first dimension (Spots) and then for each spot over its second dimension (Instruments and their parameters that have to be changed).

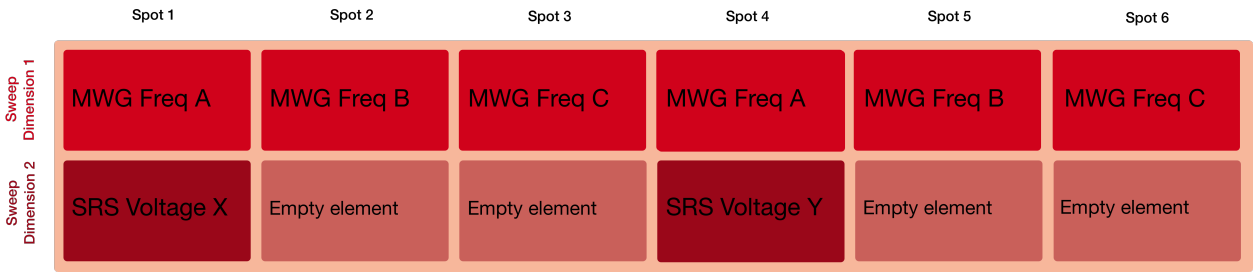


Figure 4.10: An example for the *Spot Array* of a 2D sweep. Its first dimension (horizontal) runs over the spots, its second dimension (vertical) over the sweep dimensions (here: "Frequency" and "DC Voltage").

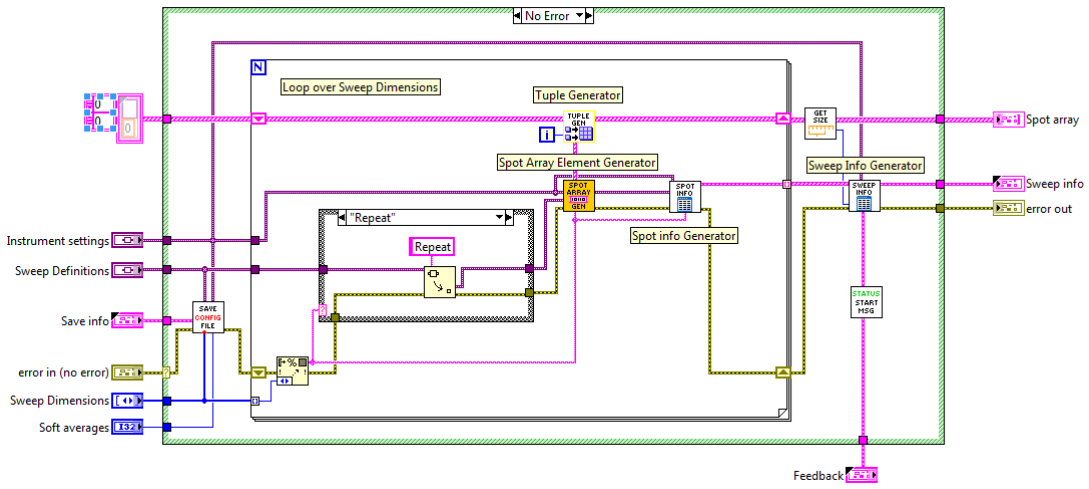


Figure 4.11: The LabView block diagram of *SweepSpot generator*. The big for-loop in the middle runs over the *Sweep dimensions*. In the VI *Spot array generator* the spot array of the current dimension is created. This is combined with the main spot array in *Tuple generator*. Additionally, metadata about the experiment is collected in *Spot Info Generator* and *Sweep Info Generator*.

The *Spot array* is built by *SweepSpot generator* from the defining settings of the experiment as follows (see Figure 4.11): First, *SweepSpot generator* loops over the *Sweep dimensions*. It begins with the fastest dimension. First, it takes the corresponding element of *Sweep Definitions*. If we again consider the example of Section 4.1.2, this would be a frequency sweep. Such an element usually defines the range and a step size of the sweep. *Spot array generator* then builds an array out of the definition of this range and step size. For simplicity it can be assumed that the sweep contains only three different frequency values, as illustrated in Figure 4.12. This array is then passed to the subVI *Tuple generator*. For the first dimension, this VI does nothing meaningful to the *Spot array*. However, *Tuple generator* will become important when the second dimension will be added. Additionally, there is the *Spot info* array that contains metadata for each sweep dimension. Usually this is the definition of the sweep, i.e. start point, stop point and the step size. This information is added to the *Spot array* in *Spot Info generator*.

Figure 4.12: The raw spot array for the first dimension, i.e. a Frequency Sweep.

Now the main loop in *SweepSpot Generator* is repeated for its second iteration. Again the corresponding element of *Sweep definitions* is taken and a spot array of the current sweep dimension is created by *Spot array generator*. Now, *Tuple generator* has to merge the two spot arrays of the individual dimensions. As Figure 4.13 shows, *Tuple generator* combines the two arrays in such a way that there are as many elements in each spot as there are instruments to be set. If one of the sweep dimensions does not have to be updated, an empty element is added. This way it is ensured that SweepSpot does not unnecessarily communicate with instruments as this takes time. Furthermore, the metadata for the current sweep dimension is added to the *Spot info* array just like before. After having iterated over all sweep dimensions, the VI additionally builds the *Sweep*

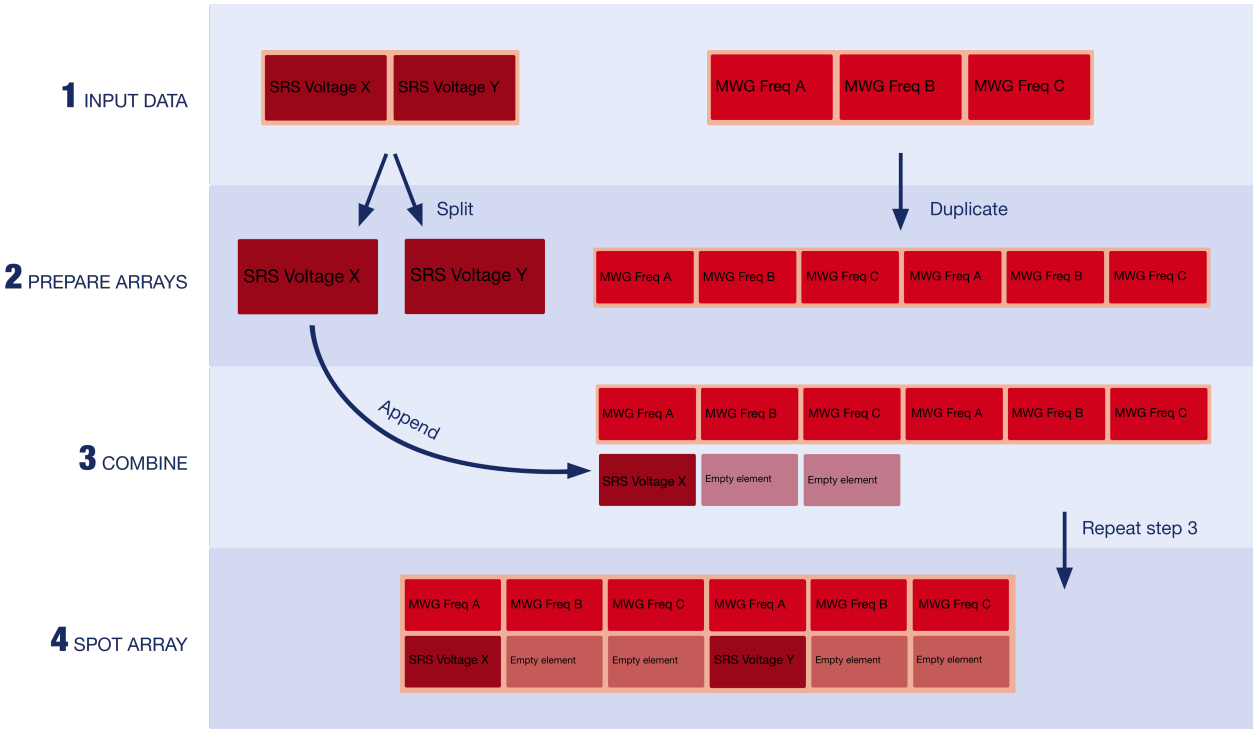


Figure 4.13: A schematic overview of *Tuple Generator.vi*. First the current spot array is duplicated as many times as needed. Then the new elements are appended to this duplicated array.

info cluster. This contains additional "global" information about the experiment such as the start time, the length of the data of the fastest sweep dimension or the acquisition devices that are used. The *Spot array* with the metadata about the individual sweep dimensions is then added to *Sweep info* as a subelement. Most of the information in *Sweep info* will later be added to the header of the data files. Some of the information will also be used to generate plots in *SweepSpot frontend*.

In the end, *SweepSpot generator* has created two arrays: the *Spot array* that contains information about instrument settings that have to be changed in each spot and *Sweep info* that contains metadata about the experiment. Additionally, *SweepSpot generator* writes and saves the frontend configuration file which contains all the settings of the current experiment. More about that can be found in Section 4.1.5. Now SweepSpot is ready to begin with the actual measurements. This is the subject of the next section.

4.1.4 SweepSpot.vi

SweepSpot.vi is the second subVI of *SweepSpot main* and the one that performs the measurements. As Figure 4.14 shows it is built up of three parts: *Prepare sweep* that sets the default settings, a loop over *Spot.vi* where the measurement is performed and *Motion reversal* that cleans up the software part of the setup.

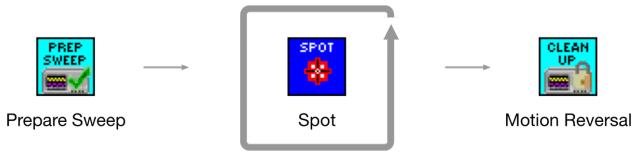


Figure 4.14: Schematic representation of *SweepSpot.vi*. First the instruments are prepared, then for each spot a measurement is performed and in the end the setup is cleaned up.

Prepare sweep

As depicted in Figure 4.15, *Prepare sweep* first initializes the instruments that will be used in the experiment. This is done by the subVI *Initialize instruments*. The VI connects to the devices and ensures they are ready for the experiment.
Set instrument afterwards sets the initial parameters of those devices. These are the values defined in *Instrument settings*. *Set instrument* loops over all possible device types (e.g. "AGW", "MWG", "FPGA", ...) and looks for corresponding elements in the data input array. If it finds a corresponding element it sets the new values using subVIs that are unique to each device type. Furthermore, the *Measurement data* array is created with the correct size. Like this LabView only allocates memory for *Measurement Data* once which makes SweepSpot more time efficient. Now that all initial parameters are set the sweeps can begin. Next, *SweepSpot.vi* calls *Spot.vi* for each intermediate step of the experiment. In particular, *Spot.vi* is called as many times as there are spots in the experiment. Additionally the whole sequence is repeated as many times as specified by the number of soft (=software) averages. SweepSpot then averages over the different datasets.

Figure 4.15: The basic structure of *Prepare Sweep.vi*.

Spot.vi

The *Spot* VI, schematically shown in Figure 4.16, handles all tasks that have to be repeated in every spot. This includes the setting of the new instrument parameters to the data acquisition and frontend feedback handling.

Figure 4.16: A schematic representation of *Spot.vi* which is the main subroutine of *SweepSpot.vi*.

In each spot the software performs the following steps:

1. The instrument parameters for the current spot have to be updated. This is done by the *Set instrument* VI. The VI loops over the remaining dimension of the current *Spot array* element. This is the list of changes for each instrument type that are required for the current spot.
2. Then in *Make acquisition ready* all enabled acquisition devices are set to a "ready-to-measure" state.

3. *Start devices* makes sure all relevant AWGs are running. Some AWGs may have been stopped to prevent sending triggers to other instruments, in particular to the FPGA. In this step these AWGs are started again if needed. Additionally, if the "AWG Resync" option is enabled, this VI stops all AWGs. This sets the first waveform of the sequence to be output at the next trigger. Then, the AWGs are switched on, with the Master AWG that triggers the others as the last one. Like this the waveform counters of the AWGs are synchronized.
4. The readout is done in *Readout acquisition*. There the acquisition devices start measuring and append the new data to the former measurement dataset.

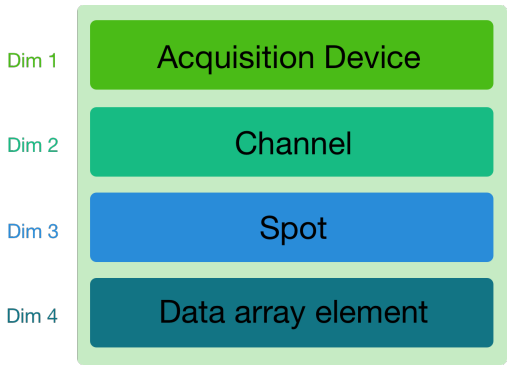


Figure 4.17: The four dimensions of the *Measurement Data* array.

The acquired data is stored in a 4D array called *Measurement data* (see Figure 4.17). The first dimension runs over the list of all acquisition devices that were enabled. The second dimension specifies the channel number of one particular acquisition device. The third dimension runs over all spots *for data coming from a particular channel of a particular acquisition device*. The actual measurement data of this spot is then in general one-dimensional. This 1D array is what the fourth dimension stands for. As an example one can consider taking data with an FPGA. The FPGA always records a vector of data points for each spot, i.e. a 1D array of data points. The fourth dimension runs over this 1D array. The list can also consist of only one element (e.g. for the FPGA option "Average over time") or it can be a flattened list of 2D data which will later be unflattened again. This fundamental structure of how to handle measured data should also be followed when implementing new acquisition devices, as explained in Appendix B.4.
For more insight to the *Readout acquisition* VI it will now be shown how SweepSpot handles different kind of FPGA data:

- **One Sweep Dimension "Frequency 1", "Average over time" option disabled**
  This is the simplest case. Here the FPGA measures a time trace for each spot. We assume that the FPGA parameter "Number of segments" is 1. This means that time trace data is not further processed. A bit in more detail: The FPGA outputs a 2D data array but as the number of segments is one only the first slice carries data; this is the time trace. The other array entries are empty.
- **One Sweep Dimension "Frequency 1", "Average over time" option enabled**
  Here the FPGA as usual measures a time trace for each spot. Right after the data is passed to SweepSpot, LabView averages over the 1D data array. This results in a single

point for each spot, i.e. a 1D data array with only one entry.

- **Two Sweep Dimensions: "Frequency 1" and "DC Voltage", "Average over time" option enabled**
  This case is similar to the previous one. Generally there are more datapoints as there are now two sweep dimensions. But for each spot one still gets only one datapoint which is the average over the time trace of the current spot.
- **One Sweep Dimension: "AWG sequence" with one sequencefile loaded. "Average over time" option disabled**
  This example shows how SweepSpot handles data for an experiment where the number of FPGA segments is bigger than 1. Here the FPGA delivers 2D data, i.e. a 2D array of data with as many rows as there are segments. In case of enabled "Average over time" option the length of the rows is just 1. SweepSpot now flattens this 2D matrix to a 1D array to make it fit into the general structure of the data. As depicted in Figure 4.18, SweepSpot basically treats each segment as an additional spot. In the end there are as many spot entries in *Measurement data* as there are FPGA segments, each containing its 1D time trace data. To keep the concept of the Mathematica analysis files this data is unflattened again to 2D data in a subVI of *Save measurement data.vi*.

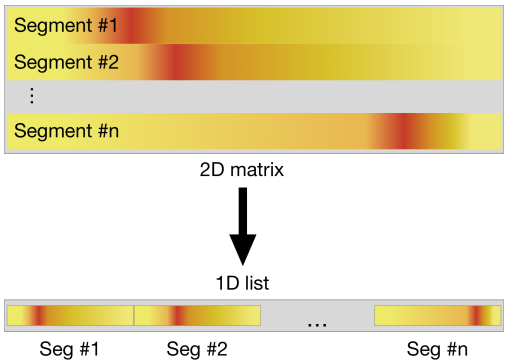


Figure 4.18: 2D data of an FPGA is flattened to a 1D array.

Let us finally go back to *Spot.vi* and look at its last subVI (see again Figure 4.16).

5. Finally there is the *Feedback* VI that sends information about the status of the experiment back to a frontend. It updates a frontend status box with the information about the current spot index and the estimated remaining time. It also updates a frontend progress bar. Moreover it sends an "Update" notifier to a frontend stating that new data has been acquired.

Motion reversal

Finally, after all data has been acquired a proper "clean up" of the devices is needed. This includes closing all open VISA handles. In that sense *Motion reversal* ensures that all the instruments are ready for the next experiment. Now SweepSpot is ready to save the gathered data.

4.1.5 Save measurement data / file handling

Save measurement data is the last subVI of *SweepSpot main*. Its task is to write the data files and in general also the pulse configuration files. First an overview of the saved file types is given.

File handling

For each experiment SweepSpot in general writes three type of files to the file system:

- For every experiment there is a **frontend configuration** file. It has the ending ”.cfg” and a unique infix ”CFG” in the filename that is required by the Mathematica functions to identify the file. The configuration file contains all information about the experiment, i.e. all instrument parameters and some global information such as the number of soft averages and the filenumber. This file can be used to look up certain settings afterwards. It can also be loaded using the ”Load” function on *SweepSpot frontend* to redo the experiment. The structure of the file is similar to the configuration file that Cleansweep used but it contains additional instrument settings that are not implemented in Cleansweep but in SweepSpot. Also the underlying structure is slightly different (it is based on the LabView cluster *SweepSpot settings* instead of the Cleansweep datatype *Experiment settings*).
- Depending on the acquisition devices used to read out the data there are several **data files** written. SweepSpot saves at least one file (this depends on the acquisition device) per readout device with the corresponding data. The files carry the extension ”.dat”. Each acquisition device may handle its data differently. SweepSpot saves I and Q data files for each active channel, so at the most 4 files for an FPGA based measurement. The channel number and the quadrature component of the data are also added as an infix to the filename.
- Optionally, SweepSpot writes a **pulse configuration file**. This file describes the waveforms generated for the AWG. It contains basically all information of the current ”Pattern Configuration” file (a file containing information about the current sample, e.g. the IF-frequency for a sideband modulation for the qubit drive) and some additional information about the experiment, such as the index of the segment that carries the calibration information for the qubit $|e\rangle$ state. For usual spectroscopy experiments one does not need to have such a file. However, if the data is analysed using the automated Mathematica analysis functions in ”CalibrateAll.m” such a file is required. More importantly, the file can be used to read information about the current settings of the experiment at a later stage. SweepSpot saves a pulse configuration file if it detects a nonempty path in AWG settings $\rightarrow$ Sequencefile path. This default behavior can be overwritten by specifying a different path in Save info $\rightarrow$ PatternConfig file. If both paths are empty SweepSpot does not write a pulse configuration file. The file carries the ending ”.dat” and the identifying infix ”Pulses”.

The Qudev Mathematica framework has been extended as part of this project such that the analysis functions can handle Cleansweep and SweepSpot data. To read in SweepSpot data one has to choose the option value *source* $\rightarrow$ *”SweepSpot”* for the function *ReadInData*. This setting can potentially also be chosen in a higher level function that calls *ReadInData* as a subroutine. The analysis functions have not only been extended to SweepSpot data but have also been generalized. As an example, there is now the function *GetExpFilenames* that finds specific files of an experiment that respect the naming convention. One has to enter the path of the files and can then choose to look for all files with a certain filenumber, for files with a particular infix or even a very specific file. These generalizations also appear at various other places in the code.

The filenames are logically structured. As shown in Table 4.1 the first part of the name is the filenumber. Then the prefix specifies what type of file it is, e.g. ”Pulses” for a pulse configuration file. Then for the data files it follows the channel number the data comes from. Finally all files carry the ”identifier” which is typically set by the user on a front end and tells something about what kind of experiment it is (e.g. ”Rabi”).

Table 4.1: SweepSpot file naming convention

Frontend config file	Data files	Pulses config file
1000_CFGRabi.cfg	1000_I2DPlotCh0Rabi.dat	1000_PulsesRabi.dat
	1000_Q2DPlotCh0Rabi.dat	

Save measurement data VI

The data files and the pulses file are written in *Save measurement data* which is depicted in Figure 4.19. The procedure goes as follows:

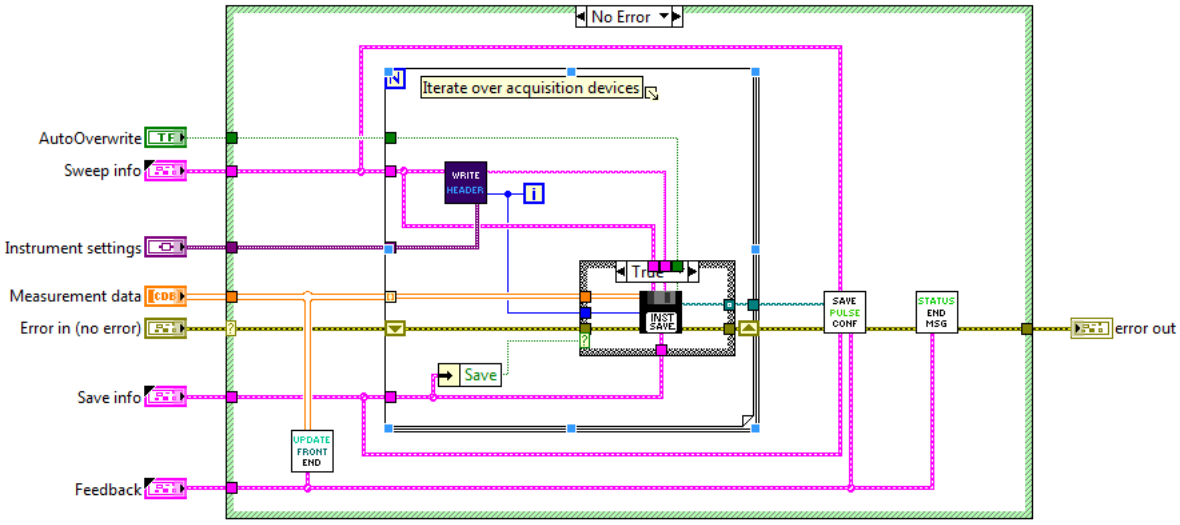


Figure 4.19: The LabView Block Diagram of the VI *Save Measurement Data*. The middle for-loop loops over the different acquisition devices that were enabled. First the header of the files is prepared and then the data is saved. After the loop (the VIs on the right side of it) the pulse configuration file is saved and the frontend is updated.

1. First, *Update frontend*, as the name suggests, makes sure that any frontend plots show the full measured data by sending an ”Update” notifier and updating the reference to *Measurement data* with the final dataset.

2. Then the actual saving procedure starts. SweepSpot first iterates over the different acquisition devices. This is the first dimension of *Measurement data* and the main loop in *Save measurement data*.
3. As a first part of one such loop iteration, SweepSpot writes the header of the data files. The header (see Figure 4.20) is formatted as a Mathematica association containing basically all the information from *Sweep info*. This includes general metadata such as the current acquisition device and the definitions of all sweep dimensions that are relevant to reconstruct the measurement points during analysis.

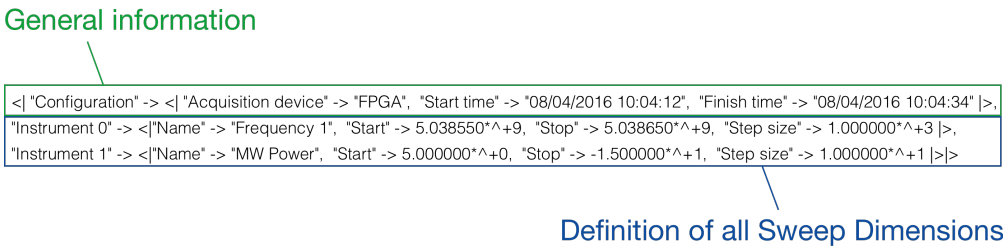


Figure 4.20: A typical header of a SweepSpot data file

4. Then *Save instrument data* writes the data files for each channel of the current readout device. At this stage the fastest dimension of *Measurement data* is unflattened if the number of segments is bigger than one. This only affects data that was originally 2D (or even higher dimensional) and that was flattened to 1D data before (see Section 4.1.4). Afterwards the header is prepended to each datafile. Like this the data files all carry the same structure as shown in Figure 4.21.

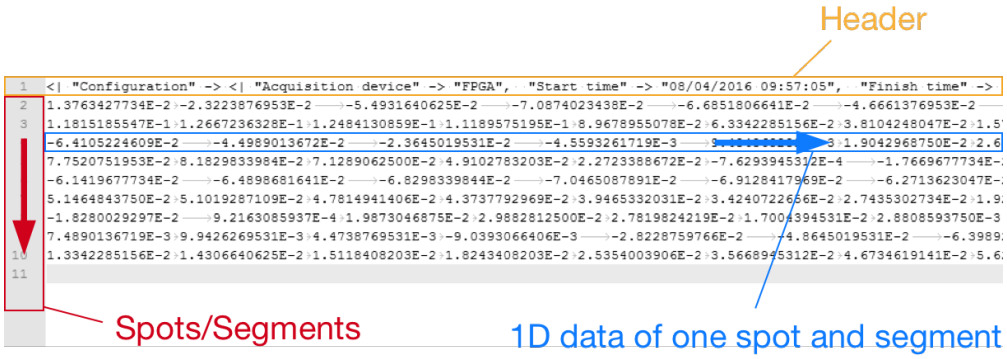


Figure 4.21: A typical SweepSpot data file.

Then the loop is repeated for the second acquisition device, and so on. There are two supported file formats: ASCII and binary. SweepSpot choses the format that is set in the *Save info* cluster.

5. Finally, the pulse configuration file is saved (if needed) and a message is sent to the frontend that the experiment has finished.

4.2 SweepSpot frontend

Sweep Spot frontend is a VI that can be used to perform experiments directly using *SweepSpot main*. If one wants to do more complicated sequences of experiments one should use higher-level VIs such as *QubitCalib* that also use *SweepSpot main* as a backend module, as explained in Chapter 5. *SweepSpot frontend* is shown in Figure 4.22. On the upper left corner there is a status box that

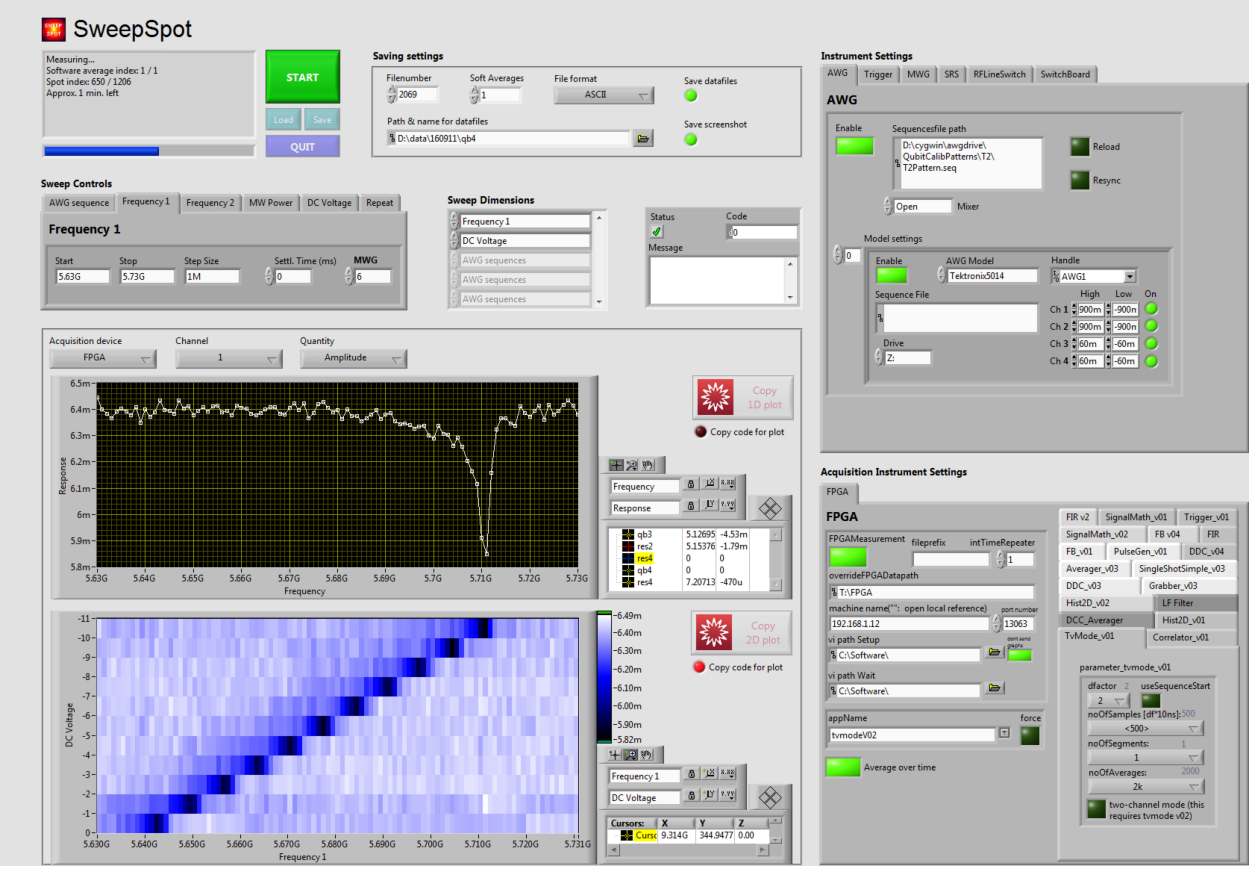


Figure 4.22: *SweepSpot frontend.vi*: The main user interface of SweepSpot. On the top there are the main controls, the experiment control buttons and the status feedback box in combination with a progress bar. In the middle there are the controls for the *Sweep Dimensions* and the *Sweep Controls* as well as a box that displays errors. On the bottom there are the 1D and 2D plots and their controls and on the right side there are the controls of the instruments.

displays information about the current state of the experiment such as the current spot number, the estimated remaining time and what SweepSpot is currently doing. The progress bar indicates how far the experiment is. Then there are 4 buttons: Start/Stop, Quit, Load and Save. The Load and Save buttons can be used to load/save a frontend configuration and with the Start/Stop button one can run or abort an experiment. The Quit button aborts the whole VI and only has to be pressed when SweepSpot is not used anymore. On the top in the middle there are settings about how and where to save the files. In the middle row one first finds the *Sweep Controls* where the individual sweeps can be defined. To actually choose the sweep types the *Sweep dimensions*

box in the middle can be used. On the right-hand side of it there is a box that displays error messages. On the bottom of the VI there are the 1D and 2D plots and their settings. On the top right there are the settings for the instruments and on the bottom right the controls for the acquisition devices.

Start an experiment

If one wants to do a measurement with *SweepSpot frontend* the following steps have to be done:

1. First the LabView VI has to be running. A VI can be started using the ”Run” button on the top left corner of the LabView toolbar. This only has to be done for the first time.
2. Now one can set up the first experiment. First, a sensible filename has to be chosen using the filename text field on the top of the frontend. The name specified here will end up in all filenames of the current experiment as a suffix (e.g. ”Rabi”). Furthermore, one can check whether the current filenumber is correct and the file format can be chosen. If one does not want to save any files at all the ”Save” button can be disabled. If one is not interested in screenshots of the frontend configuration one can disable the ”Save screenshot” button.
3. Now the actual experiment has to be defined using the *Sweep Controls* and the *Sweep dimensions* controls. One has to choose which parameters should be swept using the *Sweep dimensions menu*. For each item that is selected in this array one also has to set the corresponding parameters using the *Sweep Controls* tabs.
4. Next, one should make sure that the settings for the instruments and acquisition devices are fine. These of course depend on the experiment. If not interested in time traces while using an FPGA, the ”Average over time” button has to be enabled. SweepSpot will read out measurement data with all acquisition devices that are currently enabled.
5. To begin with the experiment one simply needs to press the big green Start button. To abort the measurement the same button which has now turned to a big red Stop button can be clicked again. The data gathered so far, as well as the configuration of the aborted experiment, are still saved.

Like in *QubitCalib frontend*, the front panel is meant to be running for the whole set of experiments. There is no need to stop the whole VI using the Quit button throughout the experiment session.

Plots in Mathematica style

One new feature of SweepSpot is that using the frontend a Mathematica plot of the experiment data can directly be copied to the clipboard. One simply has to press the ”Copy plot” button on the right-hand side of the graph. The current Mathematica version can be changed in the code diagram of *SweepSpot frontend*. Thanks to this feature the plots in the electronic labbooks will look more professional than the screenshots of the LabView graphics. However, as the plots may need to be edited first, there is an option to copy the code that generates the plot instead of the plot figure itself. Like this the copied code can be pasted to a Mathematica Notebook and edited there. This makes it possible to analyse data on the fly and display the results of the experiment in specific ways.

Load and save a frontend configuration

As explained before, SweepSpot automatically saves the current frontend settings as a ".cfg" file when an experiment is started. There is also an option to save such a file manually: one simply has to press "Save". Then one gets prompted to enter the location where the file should be stored. Similarly, such a frontend configuration file can easily be loaded to *SweepSpot frontend* to repeat a measurement. To do this one has to press the "Load" button and then choose the frontend configuration file. This corresponds to the "Quick Switch" function in Cleansweep. *SweepSpot frontend* can read configuration files that were written by SweepSpot or Cleansweep even though the two files are formatted differently. However, if one uses Cleansweep frontend configuration files there may be settings that are not set in *SweepSpot frontend* like the controls for the DC sources as these are not implemented in Cleansweep.

4.3 Efficiency of time and memory usage

The LabView code of SweepSpot was optimized in terms of time efficiency and memory usage to a large extent. It was ensured that as little memory as possible is de- and reallocated to prevent a slowdown of the application. Furthermore it was a goal to only use little memory in general to prevent a memory overflow. For that the underlying data types were carefully chosen and time consuming subVIs are ensured to only be executed when absolutely needed. Thanks to these optimizations SweepSpot outperforms Cleansweep in terms of time efficiency. As shown in Figure 4.23 (a), SweepSpot can perform the same experiment 15% faster if the experiment is short (about 10 seconds). For long experiments it outperforms Cleansweep by up to 5% execution time. For a measurement of a few hours this makes a significant difference. The time efficiency is bigger for short measurements as SweepSpot performs the preparation steps of an experiment more efficiently. For short experiments this improvement has a much bigger influence on the total time.

Figure 4.23 (b) shows the time distribution of the different main steps of an experiment with SweepSpot. The four main contributions to the total time are the time during that the experiment spot takes place, the time to set up the FPGA before each of the measurements, the time to communicate with the other instruments (such as AWGs, MWGs) and the time for LabView to execute the code (including plotting, frontend updates etc.). For a small number of averages, as shown in Figure 4.23 (b), the FPGA setup takes most of the time. In this step the memory of the FPGA is cleared for the next experiment. It should be investigated if the FPGA could do this step more efficiently. If so this would result in a significant decrease of the total experiment time. Raising the number of spots increases the time spent by the FPGA setup. This is not fully understood and should be investigated further. Moreover the percentage spent on executing the software itself decreases for a higher number of spots. The reason is the same as already mentioned above: the time for the initialization and preparation of the experiment is now averaged over a higher number of spots which decreases the contribution of code execution.
The statistics presented in Figure 4.23 strongly depend on the exact configuration of an experiment as there are many factors that influences the overall time of an experiment. The corresponding settings of the test experiments shown in this chapter can be found in Appendix A. First of all, the AWG pattern, the main repetition rate as well as the FPGA settings determine how long a

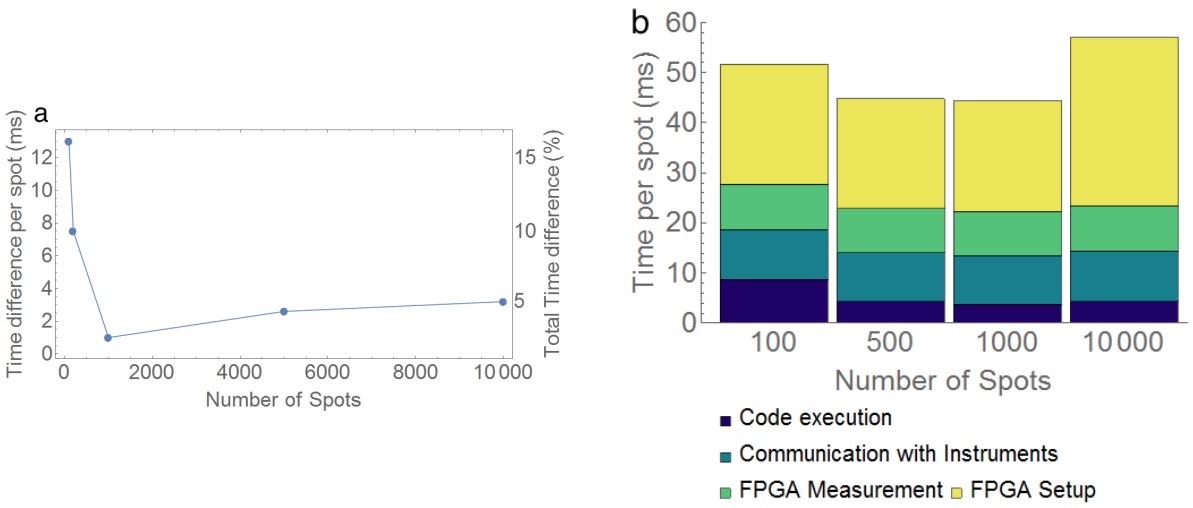


Figure 4.23: (a) The time difference between performing an experiment with SweepSpot and to performing it with Cleansweep. SweepSpot outperforms Cleansweep by 15% of the total time of an experiment for short measurements and by around 5% for long experiments. (b) Time statistics for an experiment with SweepSpot performed with a different number of spots. The four main contributions to the experiment time are FPGA setup, FPGA measurement, communication with instruments and code execution. The FPGA setup takes the most time for a low number of averages (1k). The detailed configuration of these experiments are described in Appendix A.

measurement takes. Then, the number of spots and software averages determine the total time of the whole experiment. Also the settling time has an influence on the total execution time. After the setting of the instruments in the beginning of a spot, SweepSpot pauses the execution for the time defined by the settling time control. For a multidimensional sweep with different settling times defined, SweepSpot takes the highest one for a given spot. Moreover, there are two subtle factors that also have an influence on the total experiment time. First, a large number of enabled instruments on the SweepSpot frontend may slow down the experiment as the communication between LabView and an instrument takes time. Each instrument that is enabled is in general only accessed once in the beginning and once in the end of the experiment. However, in each spot SweepSpot communicates with *all enabled microwave generators and all enabled acquisition devices*. Therefore to speed up the software instruments that are not relevant should be disabled. Second, LabView by default continuously updates every open Front Panel throughout the experiment which takes a significant amount of time. While having the *SweepSpot frontend* window open makes sense, further subVIs should only be open if absolutely necessary.
For the experimenter to improve the time efficiency of the experiment SweepSpot shows a feedback on the frontend status box stating how efficient the current experiment is in terms of execution time. The time efficiency is defined as the percentage of the whole experiment time that is spent on measurement:

$$\text{time efficiency} = 100 * \frac{\text{measurement time}}{\text{total time}} \tag{4.1}$$

A high time efficiency states that that a good fraction of the overall time is spend by measuring data. On the other hand a low time efficiency hints that there is some significant amount of

time SweepSpot does other things than measuring. In particular, there could be some dead time where the FPGA waits for a trigger to start measuring, e.g. to wait for the qubit to decay; there could be a lot of enabled instruments that need to be set in each spot; the FPGA TvMode_v01 settings could not be optimized for the current pattern in terms of time efficiency; there could be lots of open LabView Front Panels in the background; or there could be some other issue. The efficiency indicator on the frontend should force the experimenter to think about optimizing the time efficiency of an experiment - in particular if it is a very long one where time efficiency has a big effect.

4.4 Improvements and differences to Cleansweep

One of the main advantages of the SweepSpot framework compared to Cleansweep is that its code is designed to be cleaner and more logically structured. This should improve code readability and in that way also the process of adding new features. This not only holds for the code itself but also for the descriptions on the LabView VIs, their conceptual icons and the explanations of the software in this thesis and on the Qudev-Wiki. Overall the SweepSpot framework also provides a more adaptable environment to implement new functions.
Moreover, as discussed in Section 4.3, the software execution time overhead was reduced.
Third, SweepSpot allows multidimensional sweeps. The user is therefore not limited anymore to 2D sweeps but can use an arbitrary number of sweep dimensions.
Furthermore, there were various improvements made in the Mathematica framework that is used to read out and analyse the data written by SweepSpot or Cleansweep. Several functions that has been restricted to certain parameters have been generalized and cleaned up.
Then, there are already instruments that are only implemented in SweepSpot, such as a DC source or the automation of the movement of piezoelectric elements in a setup with a laser microscope. However, there are also still some features missing such as the implementation of the Acquiris and Zurich Instrument devices.
In addition, *SweepSpot frontend* was designed to provide a clean, user-friendly interface to make it as intuitive as possible to perform experiments. The software provides status updates during the experiment that show how far the experiment is and what the software is doing. Opposed to Cleansweep, the frontend also comes with a progress bar that also tracks the fastest dimension. Furthermore the remaining time and the time efficiency is calculated. Moreover, the feature to automatically generate a Mathematica plot of the gathered data is thought to contribute to a professional documentation of the experiments.
And finally, the concept of using the VIs as sequentially reusable modules is crucial for the construction of higher-level VIs that automize spectroscopy. This will be seen in the next chapter which describes how the SweepSpot framework can be used for more complicated functions and how the new software suit opens up to completely new possibilities.

Chapter 5

Spectroscopy Automation

As explained in the previous chapter, the SweepSpot framework can easily be used to build more complex functions such as for the automation of spectroscopy experiments. In this chapter different LabView/Mathematica functions will be presented which for the first time automize spectroscopy experiments. The routines have been developed as part of this project and their implementation only got possible using the new SweepSpot framework. But first it will be shown how SweepSpot fits into the calibration automization software framework that is used in the Qudev laboratory to perform more advanced experiments.

5.1 Software overview

The heart of the calibration automatization software framework in the Qudev laboratory is a LabView application called *QubitCalib*. It is a program that runs a sequence of experiments and automatically analyses their results. A typical characterization procedure of a qubit is the sequence of the following experiments: Rabi, Ramsey, QScale, Rabi, CalTom and T1 and T2 measurements.

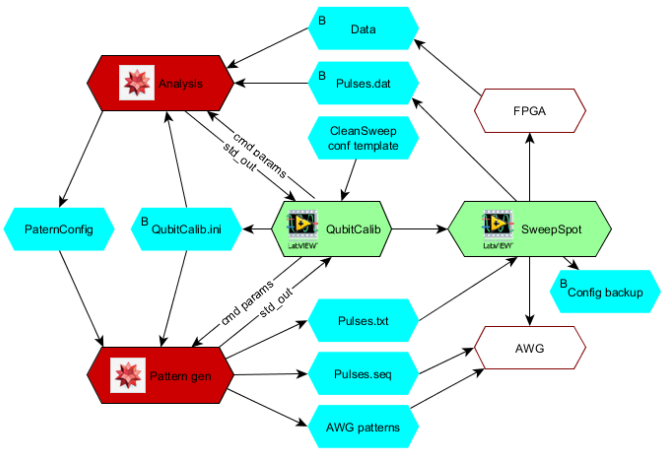


Figure 5.1: A schematic overview of the general software framework used in the Qudev laboratories. Figure adapted from [27].

For each experiment in this list *QubitCalib* calls *SweepSpot main* (formerly *Cleansweep*) to actually perform the experiment. On the level of *QubitCalib* the settings for the experiments such as the AWG sequence patterns are prepared and the data gathered by SweepSpot is analysed. The procedure depicted in Figure 5.1 goes as follows:

1. First, the user defines the sequence of experiments on the frontend of *QubitCalib* and starts the execution.
2. Then, *QubitCalib* takes the first experiment of the user-defined list and writes a initializing file (".ini") that contains all relevant information about the current experiment. This includes the type of the experiment, the path to the relevant settings files, information about which AWG channels and which microwave generators are used and general settings such as the filenumber.
3. In the next step *QubitCalib* calls the Mathematica script "GeneratePattern.m" that loads this initializing file and generates a AWG sequence pattern that will be loaded to the main AWG. It may also need to read some parameters from the "pattern configuration file" (a text document containing general information about the setup, e.g. the current qubit frequencies or $E_{j,max}$ values).
4. Now *QubitCalib* calls *SweepSpot main* or another subVI that is based on that. Among other things, the path to the generated sequence files are passed to *SweepSpot main* where the patterns are loaded to the corresponding AWGs. Now SweepSpot performs the measurement and saves the data files as explained in Chapter 4.
5. Afterwards, *QubitCalib* accesses the data and directly analyses it. This is done in the subVI *AnalyzePattern* by calling the Mathematica script "AnalyzeCalib.m" with the current initializing file. Mathematica reads the data files and the pulse configuration files that have been saved by *SweepSpot main*. The Mathematica script also updates the pattern configuration file if needed and shows a plot with the results of the experiment on the frontend of *QubitCalib*. For the next iteration *QubitCalib* repeats the procedure with the next experiment in the operation sequence list on its frontend.

Whether *QubitCalib* should call *SweepSpot main* or *Cleansweep* as the underlying software can easily be changed on the *QubitCalib* frontend on the tab "QubitCalib Configuration". Depending on the underlying module used one has to set a frontend configuration file on the *QubitCalib frontend* that was generated by the corresponding software. The operation "rebias on request" still requires Cleansweep. On the other hand there is a set of new spectroscopy routines that are only based on the SweepSpot framework (see Figure 5.4). The new operations that *QubitCalib* supports using the SweepSpot framework are:

- resonator spectroscopy;
- qubit spectroscopy;
- park qubits;
- track qubits.

These methods will now be presented in detail.

5.2 Spectroscopy.vi

Before the mentioned routines can be understood, *Spectroscopy.vi* has to be introduced. Every time some spectroscopy experiment is performed from a higher level this VI is called at some point. *Spectroscopy.vi* is a VI on a higher abstraction layer than *SweepSpot main*, in particular it

calls *SweepSpot main* as a subroutine in a loop as illustrated in Figure 5.3. Figure 5.4 shows that *Spectroscopy.vi* is the connection between all automated spectroscopy functions and the low-level *SweepSpot main*. The higher-level functions prepare the data whereas the lower level functions perform the actual experiment with the given input parameters. *Spectroscopy* takes as an input a list of frequency sweeps that has to be performed. For each of them it prepares the input parameters for SweepSpot and calls *SweepSpot main* to perform the experiment.
To embed *Spectroscopy.vi* to another VI one has to wire the following inputs:

- **Instrument settings**
  This type was already explained in detail in the previous chapter. It carries all the information about the instruments that are active during the experiment.
- **Save info**
  Similarly to the explanation in Appendix B.5 one also needs to specify where and how the data files should be saved.
- **Spectroscopy parameters**
  This is an array of so-called spectroscopy elements. A spectroscopy element, shown in Figure 5.2, is similar to the Sweep Definition "Frequency 1" (see Section 4.1.2) but in addition it carries the information about what other MWGs have to be accessed.

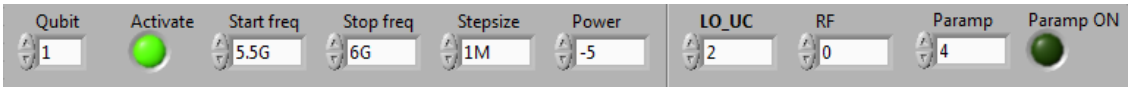


Figure 5.2: The spectroscopy element for a qubit spectroscopy experiment. The element for resonator spectroscopy looks similar but without the upconversion LO element.

- **Soft averages (Optional)**
  This integer specifies how often each experiment should be averaged over on the software side. Default: 1.
- **Feedback (Optional)**
  If a frontend is used this datatype carries references to frontend objects such as a status box that can be updated throughout the experiment.
- **Error (Optional)**
  This are the standard error input/output wires that catch errors.

5.3 Resonator/Qubit spectroscopy

The most simple spectroscopy routines are functions that look for the eigenfrequencies of a resonator or the $|g\rangle \leftrightarrow |e\rangle$ transition frequency of a qubit. As explained in Section 6.2, the two methods are conceptually similar. However, they access different parameters in the pattern configuration file to update the measured frequencies and they in general use different Mathematica fitting functions to find the optimal frequency from the measured data. The model can be chosen using the option "FittingModel". Currently, five models are implemented:

- **Single Lorentzian**
  This model is based on a single Lorentzian resonance on a flat background:

$$A \frac{|\Gamma|}{|\Gamma + i(\omega - \omega_0)|} + b \tag{5.1}$$

A denotes the amplitude of the resonance and Γ its linewidth. ω_0 is the resonance frequency.

- **Double Lorentzian**
  The double Lorentzian model takes into account other qubits/resonators that are close to the resonance that is fitted. Their effect is quantified by the addition of a second Lorentzian term to the single Lorentzian model:

$$A_1 \frac{\frac{\Gamma_1}{2}}{i(\omega - \omega_0) + \frac{1}{2}\Gamma_1} + A_2 \frac{\frac{\Gamma_2}{2}}{i(\nu - \nu_0) + \frac{1}{2}\Gamma_2} \tag{5.2}$$

  A_i are the amplitudes of the two resonances, Γ_i their linewidth, ω_0 and ν_0 the resonance frequencies of the two peaks.
- **Fano**
  This model fits a Fano resonance to the data. It is especially useful for resonances that appear as dips in a bigger Purcell filter resonance:

$$\frac{T_0}{(1+q^2)} \frac{(2Q(f-f_0)+qf_0)^2}{f_0^2 + 4Q^2(f-f_0)^2)} \tag{5.3}$$

  T_0, and q are scaling parameters of the resonance, Q is the linewidth and f_0 is the resonance frequency. This model deals with a non-flat background.
- **FanoExtra**
  In contrast to the Fano model above, the FanoExtra fit function assumes a flat background:

$$A \frac{(\frac{q\Gamma}{2} + \omega - \omega_0)^2}{(\frac{\Gamma}{2})^2 + (\omega - \omega_0)^2} \tag{5.4}$$

  Again, A is the amplitude of the resonance, Γ its linewidth, q a scaling factor and ω_0 the resonance frequency.
- **Minimum**
  This simple model looks for the minimum in a set of points.

The LabView VIs *Get resonator frequency* and *Get qubit frequency* perform the spectroscopy experiment and analyse the data, i.e. they calculate the fit function and find the optimal resonance frequency. However, *QubitCalib* calls Mathematica for the analysis which is why *QubitCalib* calls a slightly different subfunction which only performs the experiment but does not analyse the data at this stage. The analysis is done afterwards in the QubitCalib subVI *AnalyzePattern.vi*. Working with the two routines one can make use of the additional parameters on the *QubitCalib frontend*. The sweep range can be defined with the additional parameter box using the keyword "SweepRange". As shown below one should enter the Mathematica command Range[...] with the start frequency as the first element, the stop frequency as the second and the stepsize as the third, everything in GHz units.

```
{SweepRange -> Range[5.5,6,0.001]}
```

If no range is given or if the SweepRange keyword is set to *Automatic* the Mathematica routines automatically choose the sweep range. Then the current qubit (resonator) frequency set in the pattern configuration file is taken as a reference. The next sweep will now be centered around

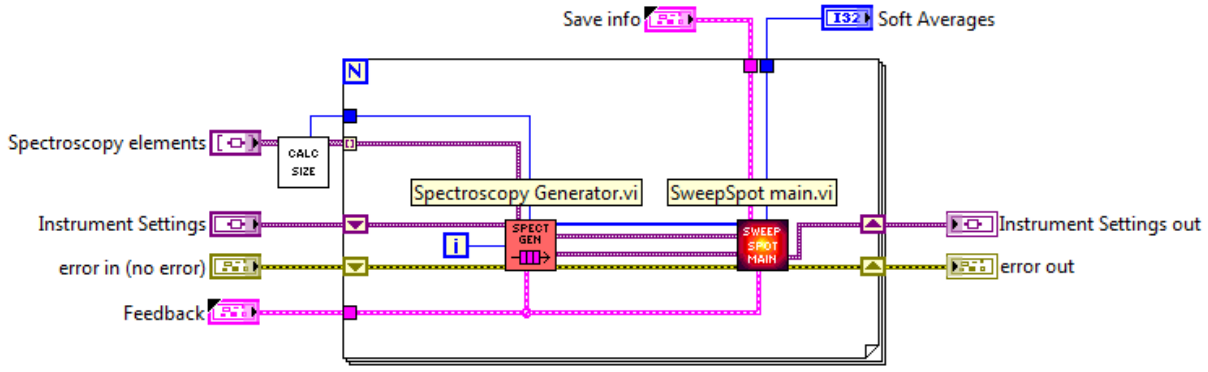


Figure 5.3: The block diagram of the *Spectroscopy* VI. It is a for-loop that goes over spectroscopy experiments that are about to be performed. For each experiment the input data for *SweepSpot main* is prepared. Then *SweepSpot main* is called to do the experiment.

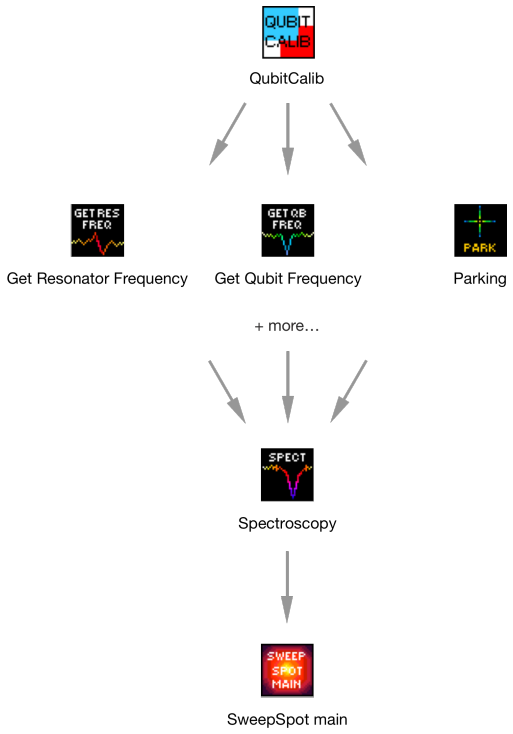


Figure 5.4: A schematic overview of the QubitCalib spectroscopy functions. The VIs on the top call the VIs on the bottom.

that frequency. The sweep range is defined by the pattern configuration parameter ”qubitSweepRange” (”resonatorSweepRange” respectively). The first parameter in the list for the current qubit (resonator) specifies the half range of the sweep whereas the second entry sets the step-

size. The entries are expected to be given in GHz units. Additionally, one can tell SweepSpot to load pattern files for pulsed spectroscopy to an AWG. To do so, using the additional parameters textbox the keyword ”ReuseWaveforms” has to be set to False. By default the value is set to True such that the patterns are not reloaded. Furthermore one can optionally set the power of the resonator that is swept using the keyword ”Power” (units: dBm). In total the additional parameters box should look similar to the following example:

```
{SweepRange -> Automatic, Power -> "16", ReuseWaveforms -> True}
```

Table 5.1 provides an overview of all possible options for the operations QubitSpec and ResonatorSpec.

Table 5.1: Keywords for ResonatorSpec and QubitSpec operations.

Keyword	**Default value**	**Further options**
SweepRange	Automatic	Range[5,6,0.001]
Power	Power of MWG set in *Instrument Settings*	”16” (any value)
ReuseWaveforms	True	False
FittingModel	”DoubleLorentzian”	”SingleLorentzian”, ”Fano”, ”FanoExtra”, ”Minimum”

5.4 Combined Spectroscopy

For every readout of a qubit transition frequency one first has to ensure that the MWG that drives the readout resonator produces microwaves with a frequency resonant to the one of the readout resonator. As this frequency shifts if a magnetic field is applied, as explained in Section 2.4, one has to look for the proper resonator frequency again before each qubit spectroscopy experiment. This can become cumbersome. *Combined Spectroscopy* solves this by performing two experiments in combination, as can be seen in Figure 5.5: the VI first scans the current eigenfrequency of the readout resonator, then it sets the corresponding MWG to this frequency to afterwards perform a qubit frequency sweep with the readout frequency set to the optimal value.
Combined Spectroscopy can be used as a module in a higher level VI. In *QubitCalib* it is equivalent to perform ”ResonatorSpec” followed by ”QubitSpec”. Like this it is ensured that the readout frequency is set to the optimal value for the qubit spectroscopy experiment.

5.5 Parking qubits

A flux model describes the correspondence between the transition frequencies of certain qubits and the applied magnetic field configuration. It is unique for every sample. If a flux model

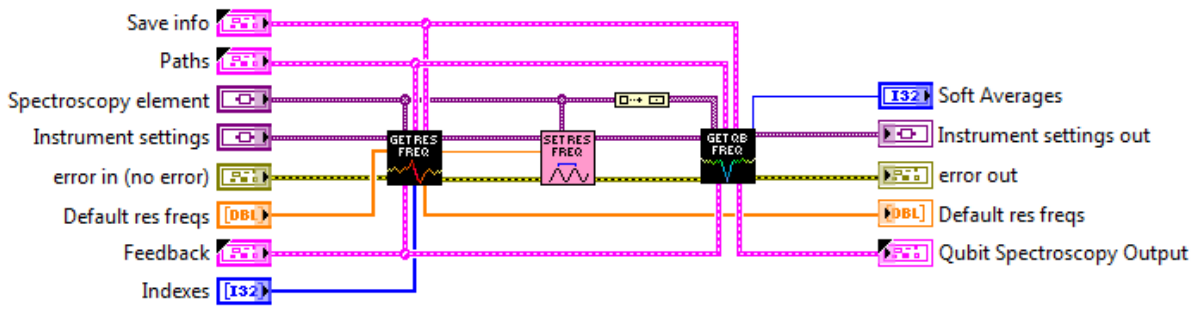


Figure 5.5: The LabView Block Diagram of the *Combined Spectroscopy* VI.

has been calculated certain qubits can be set to particular frequencies. This is what the routine "Parking" is for. A list of frequencies that the qubits should be set to serves as an input for the routine. The function then calculates the coil voltages that have to be applied to set the qubits to these particular frequencies.
The Mathematica functions are based on a flux matrix which first have to be calculated using the Mathematica function "fitFluxModel". The data for a flux matrix can be gathered with the LabView VI *FluxMatrix frontend*. This VI takes a list of coil voltages as an input. For each step it sets the coils and performs a combined spectroscopy experiment. Like this one has all the data that is needed to generate the flux matrix with Mathematica.
After converting the desired frequencies to coil voltages *Parking.vi* actually sets the coils to their new values. This is done using *SweepSpot* as a subroutine. Optionally, as a secondary test for the correct frequency values, the routine again performs a spectroscopy experiment. To do so, one has to add "Mode $\rightarrow$ Test" to the additional parameters textbox. *Parking* can be used as a standalone VI as it provides a frontend. More practically, it can be used as an operation in *QubitCalib*. For that in the operation box on the frontend one has to set two things: first, in the "Qubits" list the qubits that should be set have to be selected. Then, using "Additional parameters" the list of frequencies for the qubits have to be entered. The additional parameters should look like this (the "Test" specifier is optional):

```
Freq -> {5.1G,6.2G,7.3G}, Mode -> Test
```

The first value in the "Freq" list corresponds to the first selected qubit, the second one for the second selected qubit and so on.

5.6 Tracking qubits

As mentioned in Section 2.4 one of the features of circuit QED is that the qubit transition frequency can be changed on the fly with a magnetic field. This is interesting as the qubit does not necessarily behave equally at different frequencies. In particular the decoherence times may vary. Therefore one wants to know the characteristics of the qubit at various external magnetic field configurations. This means the external magnetic field (and in that way the qubit $|g\rangle \leftrightarrow |e\rangle$ transition frequency) is changed and at each magnetic field strength several characterizing experiments

(Rabi, Ramsey, T1, T2, ...) are performed. This process is called tracking the qubit. Unfortunately these important experiments were cumbersome to do before. In principle one could use *Cleansweep* or today *SweepSpot* to sweep the frequency in combination with the magnetic field. But this can only be done in a small range because the resonator frequency also shifts with the magnetic field. At some point the resonator is not driven at its resonant frequency anymore and therefore the qubit state cannot be detected any longer. Now thanks to the SweepSpot framework, measurements that track the qubit can be performed with ease using *QubitCalib*.

In *QubitCalib* one can generate a sequence of experiment operations. Each time one wants to change the magnetic field of the coils the operation type *SetTrackerStep* can be inserted. As additional parameters one specifies the new coil voltages like this (in Volts):

```
{"Voltages" -> {<|"Slot" -> 1, "Voltage" -> 5|>, <|"Slot" -> 3, "Voltage" -> 0|>}}
```

Alternatively, if a flux matrix is available for the current sample one can also directly enter the desired qubit frequencies:

```
Freq -> {5.1G,6.2G,7.3G}
```

There should be only as many frequency values set as there are qubits included in the flux model. After *SetTrackerStep* the next operation typically is *ResonatorSpec*. As explained above this routine looks for the readout resonators of the selected qubits and sets the new MWG frequencies in *Instrument settings*. Then one can add the *QubitSpec* operation which finds the qubit frequency and sets the frequency of the corresponding upconversion MWG in *Instrument settings* to the calculated value. There automatically the IF frequency is added such that a proper Rabi experiment can be performed. Then one can add any further operations such as Rabi, Ramsey, T1, T2 and other characterization routines, as shown in Figure 5.6 (a).
There are two big advantages of this functionality: First, it is highly practical to do tracking measurements with *QubitCalib* which already has a user friendly interface. Also using the underlying modules one can combine and reorder the subroutines. Second, one does not have to create the above sequence list for a tracker measurement by hand. An assistant was built in to *QubitCalib* (shown in Figure 5.6 (b)) that allows the user to quickly generate such a list. The assistant can be started by pressing the "Insert Tracker Element" button on *QubitCalib frontend*. Then a pop up window asks the user to insert information about the experiments that should be performed. Based on that data the assistant generates a list of operations that track the qubit.

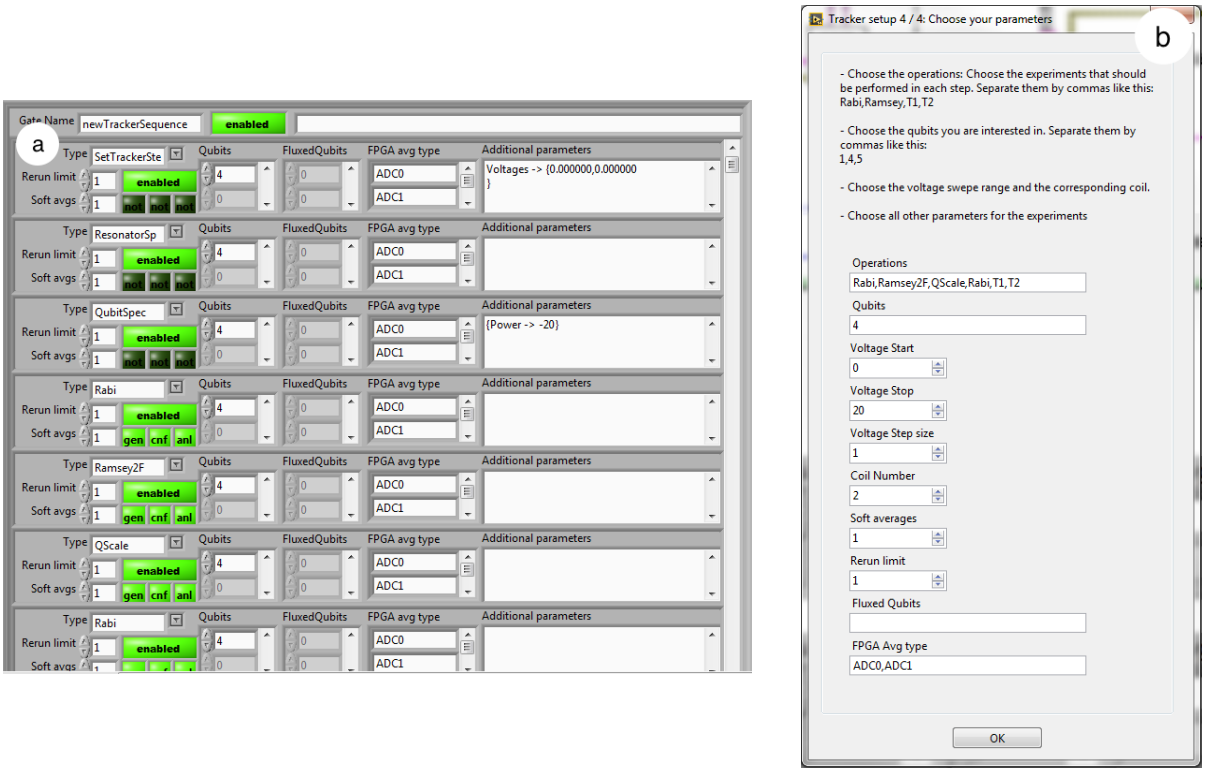


Figure 5.6: (a) Screenshot of a tracker sequence in *QubitCalib frontend*. (b) Screenshot of the software assistant that can be used to set up a tracker sequence.

Chapter 6

Sample characterization

Each quantum computing or quantum simulation experiment requires a sample with certain properties. Having designed and produced such a chip and having it built into the cryostat the first step is always to characterize it. No actual experiment can be done if the characteristic parameters of the sample such as qubit transition frequencies or coherence times are unknown. After measuring the properties of the qubits and resonators the experimenter can determine whether a certain experiment could be successful with the current chip. If the chances are high then the actual experiment can be performed, using settings that rely on the sample characteristics that were measured in this first stage.

This chapter provides a systematic approach to characterize a sample. For each step the theoretical background of the experiment will be explained, example data will be presented and it will be shown how the new software suit can be used to perform the experiments.

6.1 Sample for example data

All data that will be shown in this chapter was acquired with *SweepSpot* or *QubitCalib* running *SweepSpot* as a backend module. The data, unless noted otherwise, was gathered from the sample shown in Figure 6.1. This sample is similar in design to the one shown in Figure 2.6 apart from two differences. First, there are no flux lines on this sample. Instead, the qubit charge lines have been placed at the the top of the sample. Second, there are no coupling resonators between the qubits. They were not needed in this particular design as this sample was intended to be a test sample to investigate a new fabrication procedure of the two qubits on the right side of the chip. From now on they are denoted as qubit 3 and 4. Furthermore, the qubit design slightly differs to the one shown in Figure 2.5. Here, the SQUID loop is placed in the middle of the two islands instead of on the top. Also the resonator-qubit couplings are smaller which will have an effect when tuning the qubits with a magnetic flux bias, as explained in Section 6.2.1.
Despite these characteristics of the sample most of the steps that were used to characterize it are directly applicable to any other chip. Notable differences to the characterization procedure of a sample with coupling resonators and flux lines will be noted.

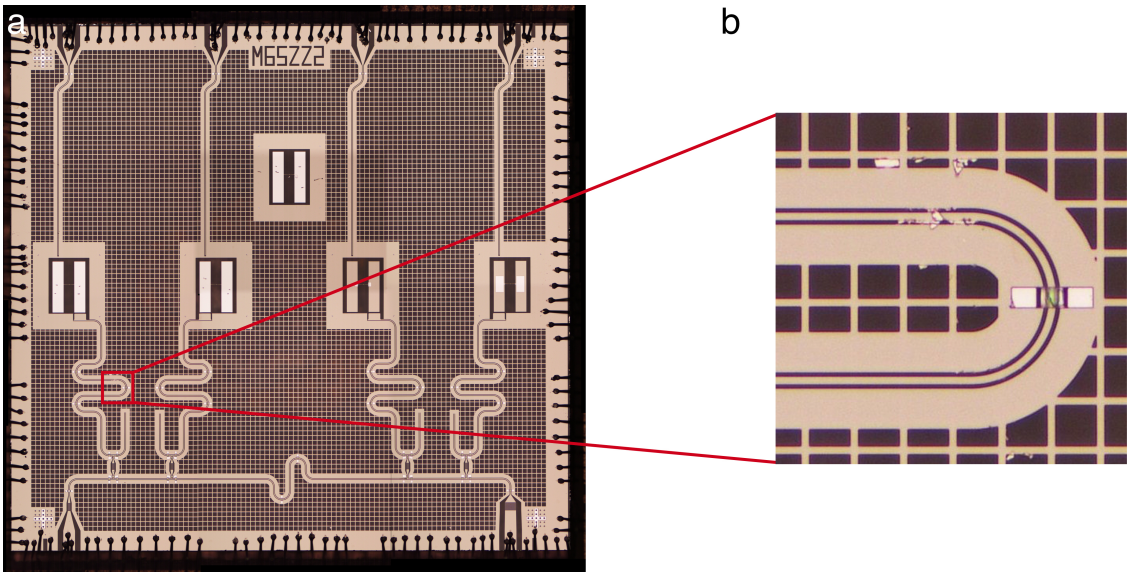


Figure 6.1: (a) A photo of the sample (M65ZZ2) that was used for the experiments in this chapter. The vertical lines on the top of the qubits are the charge lines. Each qubit on its bottom couples to its readout resonator. All readout resonators couple to a Purcell filter (horizontal line on the bottom of the sample). There are no coupling resonators on this chip.
(b) A zoom of the readout resonator of qubit 1. The transmission line is shortened by an aluminium flake that touches the ground plane (at the top of the image). Original photo by Sebastian Krinner (Qudev).

6.2 Checking the sample

As a first step it is recommended to test the general properties of the sample. In particular, we want to ensure that the resonance frequencies of all resonators are as expected and that the qubits work properly. Moreover, it is recommended to first calibrate the mixers to ensure the mixer settings are optimized for the current setup and sample. In particular, the aim is to maximize the LO suppression. This can be achieved through an iterative algorithm by measuring the quadrature imbalances and the DC biasing of a mixer [6].

6.2.1 Purcell filter spectroscopy

In a sample containing a Purcell filter the first measurement to perform is a wide spectroscopy of the Purcell filter. This allows to see whether the resonators have reasonable frequencies. In the wide spectroscopy one should be able to see all readout resonators that couple to the Purcell filter as dips in the broad Purcell resonance as shown in Figure 6.2. For the broad resonator spectroscopy a drive is applied to the input line of the Purcell filter. The frequency of the drive signal is swept over a broad frequency range. The response, measured at the output line of the Purcell filter, follows the standard spectral power density of a driven harmonic oscillator [3]:

$$P(\omega) = P_r \frac{(\frac{\kappa}{2})^2}{(\omega - \omega_r)^2 + (\frac{\kappa}{2})^2} \tag{6.1}$$

where ω_r is the bare resonator frequency, P_r the transmitted power at ω_r and κ the half width at half maximum. The lifetime of one photon in the resonator is $\frac{1}{\kappa}$. This allows to define the quality factor $Q = \frac{\omega_r}{\kappa}$ as the rate of energy loss for a unit amount of energy stored in the resonator.

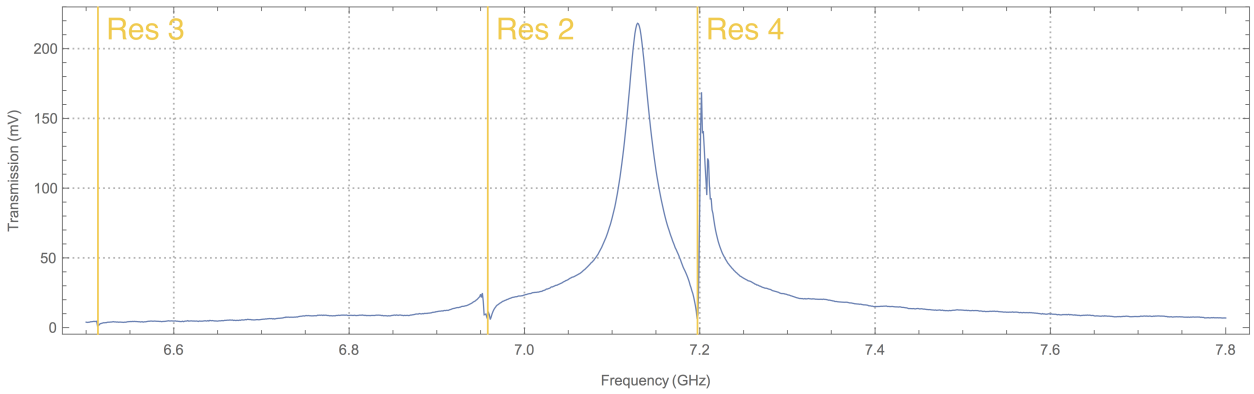


Figure 6.2: Spectroscopy of the Purcell filter. The vertical yellow lines show the bare resonances of the three readout resonators that couple to the Purcell filter.

The first result of this measurement was that the readout resonators of the M65ZZ2 sample did not have their designed frequencies. First of all, only three readout resonators could be observed. The readout resonator of qubit 1 was not seen because it got shortened by accident during the fabrication of the chip. As seen in Figure 6.1 (b), there is an aluminium flake that touches both the transmission line and the ground plane, causing the resonator to be shortened. Therefore qubit 1 could not be characterized at all. Moreover, the resonance of readout resonator 3 was shifted towards lower frequencies by a significant amount. As seen in Figure 6.2, this causes the resonator to be tiny in amplitude as it is placed in a flat region of the Purcell resonance. This fact required to drive the resonator much stronger for the spectroscopy of qubit 3. Also the bare frequency of resonator 2 was shifted from its designed position, towards the left side of the Purcell filter. Readout resonator 4 is about at its expected position. The broad spectroscopy experiment is optimally performed with no flux bias, i.e. with all coils set to 0V, as a reference point. The next step is now to look at the shifts of the resonators due to applied flux bias.

6.2.2 Magnetic field dependence

So far we have tested the resonators. Now it has to be checked whether in principle the qubits work. In particular, we want to ensure if we can see the qubit frequencies shifting when applying a magnetic field. Like this it can already be seen whether there might be a problem with some of the qubits.
For the magnetic field sweep we again do a resonator spectroscopy of the Purcell filter. Applying a magnetic flux bias to the SQUID changes E_J and therefore the qubit transition frequency (see Section 2.2). As seen in Eq. 2.17 the resonator frequency experiences a shift due to the change in the qubit frequency. In that way the shift of the qubits is detected using a resonator spectroscopy. One of the coils beneath the sample is chosen and using *SweepSpot frontend* the coil bias voltage is swept through a large range. In each step, i.e. for each flux bias setting, a Purcell resonator spectroscopy measurement as shown in Figure 6.3 is performed.

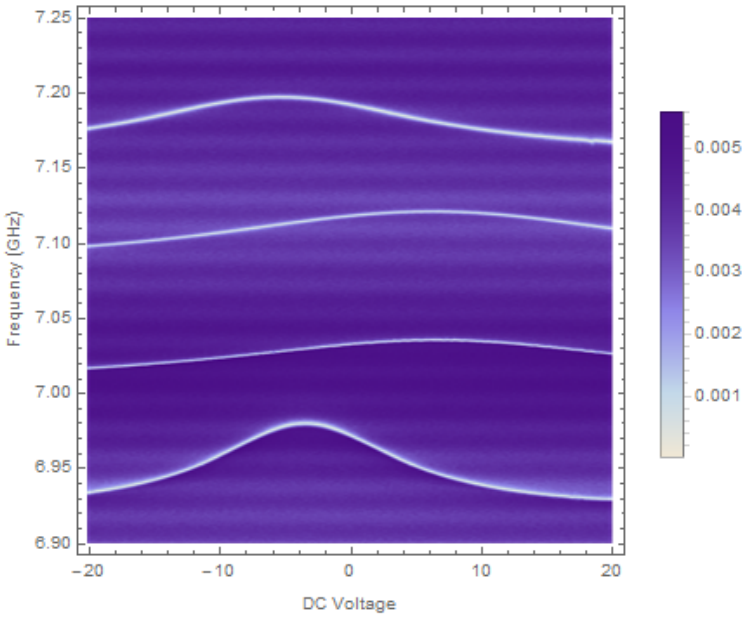


Figure 6.3: A spectroscopy of the four readout resonators at different external magnetic fields. This data was measured from the sample M65B1 which is similar in design to the one shown in Figure 2.6.

However, it is not guaranteed that one sees shifted resonators through a magnetic field sweep. Whether a qubit is tunable via the SQUID depends on the qubit design. In addition, it depends on the qubit-resonator coupling strength whether a tunable qubit induces a shift in the readout resonator frequency in a magnetic field sweep. In the case of the sample presented in Section 6.1 the coupling was designed to be weak compared to former chip designs. Therefore, one could only observe a significant shift of the resonators with a high flux bias.

Alternatively, if it is unclear whether the qubits are tunable, their drive power dependency can be investigated. In our case, as shown in Figure 6.4, we can see the static bare resonator frequency in the middle whose transmission signal disappears if populated with too few photons. For higher drive power the resonance becomes broader. This is a typical behavior of a transmitted signal in a dissipative system that follows a Lorentzian spectral distribution. However, we can also observe a nonlinear resonance in the background which is likely the shifted resonator frequency, coupled to the qubit. This nonlinearity strongly suggests that there is a qubit affecting the resonator frequency.

6.3 Qubit spectroscopy

Now one can choose one of the readout resonators to find its corresponding qubit. The following steps can be repeated for any other qubit on the same chip.

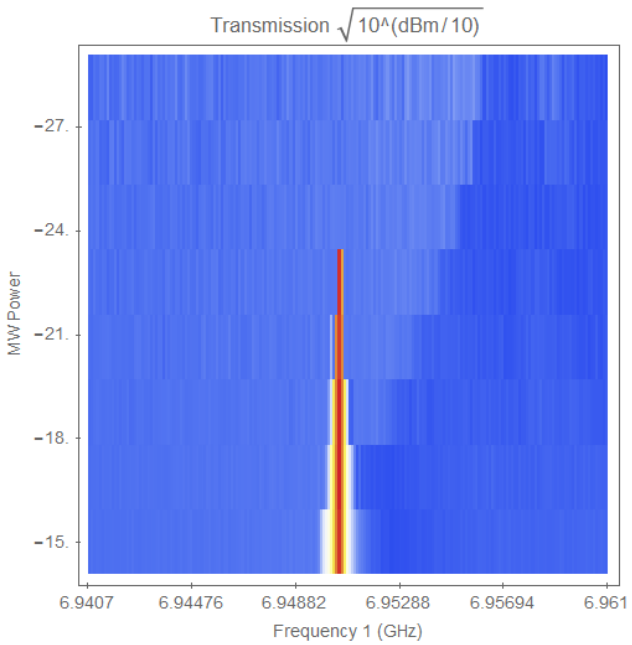


Figure 6.4: A power sweep of one of the readout resonators. The bare resonator frequency gets broadened towards higher probe power as this is the case for any transmitted signal in a dissipative system that follows a Lorentzian spectral distribution. Additionally, a nonlinearity can be observed in the background, originating from the resonator coupling to a qubit.

6.3.1 Finding the optimal readout frequency

The first thing to do when looking for a qubit is to choose the optimal readout frequency. For the experiments in this section, unless noted otherwise, it is always assumed that no flux bias is applied to the qubits. As explained before, the resonator frequency in general moves when an external magnetic field is applied. Second, the shape of its resonance changes in spectroscopy when the resonator is driven with different power, as explained in Section 6.2.2. As the aim is to maximize the transmitted signal in qubit spectroscopy, the shape of the resonator resonance is adjusted using different drive powers for the resonator, such that it will produce a high signal when shifted during qubit spectroscopy, as discussed in Section 2.4. An example of a resonator spectroscopy is shown in Figure 6.5. In general the readout resonator should be driven as weak as possible. Like this one ensures that effects due to the AC Stark shift, as explained in Section 6.3.3, are suppressed.

6.3.2 Broad and fine qubit spectroscopy

Having found a readout frequency one can now sweep the drive frequency of the qubit. It is recommended to first use high power and scan a broad range to find the approximate position of the qubit. The result of this experiment is shown in Figure 6.6. The broad peak is the $|g\rangle \leftrightarrow |e\rangle$ transition frequency whereas the narrow resonance below it is the $|g\rangle \leftrightarrow |f\rangle$ /2 transition. The next step is to find a more exact estimation of the qubit frequency by narrowing the frequency range of the sweep and lowering the power.

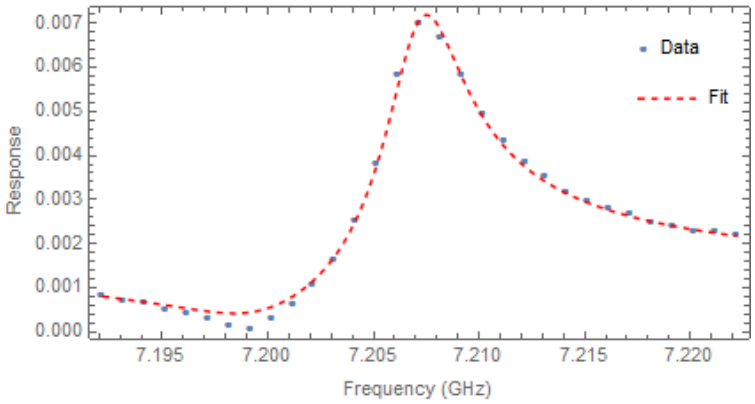


Figure 6.5: A spectroscopy of readout resonator 4, fitted with a double Lorentzian model as described in Section 5.3.

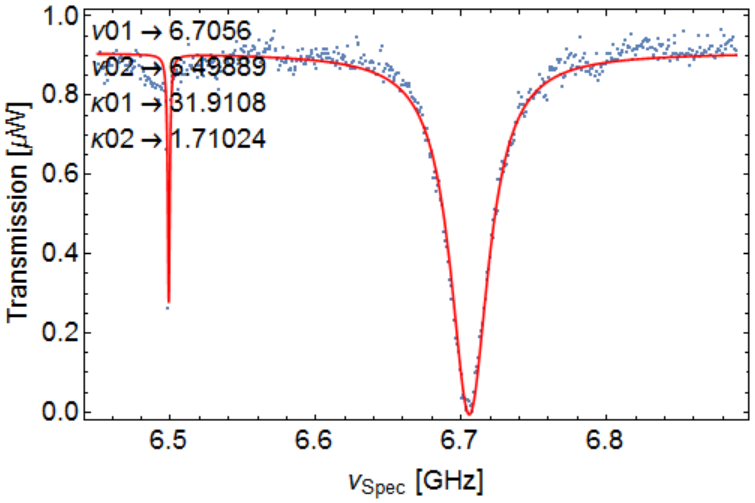


Figure 6.6: A broad qubit spectroscopy, fitted with two (uncoupled) Lorentzians. The left resonance shows the $|g\rangle \leftrightarrow |f\rangle$ /2 transition whereas the right peak corresponds to the $|g\rangle \leftrightarrow |e\rangle$ transition. Here, the qubit is driven with high power to observe the $|g\rangle \leftrightarrow |f\rangle$ /2 transition. Due to the high power the $|g\rangle \leftrightarrow |e\rangle$ resonance is strongly broadened.

The effect of the qubit drive power on the transmitted signal can be seen by the following relation of the half width at half maximum of a resonance [1], [45]

$$\delta\nu_{HWHM} \propto \sqrt{1/T_2^2 + P_{drive}T_1/T_2} \tag{6.2}$$

where $T1$ and $T2$ are the coherence times of the qubit, as described in Section 2.6, and P_{drive} is the qubit drive power. A larger drive power therefore increases the linewidth. Eq. 6.2 also shows the effect of the coherence times of the qubit on its linewidth: larger coherence times lead to narrower resonances in spectroscopy experiments. However, if the drive power is weakened

too much the $|g\rangle \leftrightarrow |e\rangle$ transition cannot be driven anymore. The reason is that the transition element $\langle g| H_I |e\rangle$ and in that way the qubit drive rate becomes small compared to the decay rate $\Gamma_1 = \frac{1}{T_1}$. Here, H_I denotes the resonator-qubit interaction Hamiltonian $H_I = \vec{d_{ge}}\vec{E}(t)$ with the qubit dipole moment $\vec{d_{ge}}$ and the electric field $\vec{E}(t)$ in the resonator. This effect directly lowers the qubit transition probability $|\langle g| H_I |e\rangle|^2$. For the same reason applying a weaker drive power makes the $|g\rangle \leftrightarrow |f\rangle$ /2 transition vanish.

Therefore, to find a more exact value for ω_{ge} the qubit drive power is reduced and the qubit is observed with a fine spectroscopy with a small range and small stepsize.

6.3.3 AC Stark shift and pulsed spectroscopy

When doing usual qubit spectroscopy caution is advised to safely identify effects due to the AC Stark shift. A resonator populated by photons induces an additional shift on the qubit transition frequency. This can be seen when revisiting Eq. 2.17. The Hamiltonian can be rewritten as [3]

$$\hat{H}/\hbar = \frac{1}{2}(\omega_{ge} + \chi_{ge} + 2\chi\hat{a}^\dagger\hat{a})\hat{\sigma}_z + \omega_r'\hat{a}^\dagger\hat{a}. \tag{6.3}$$

From here it can be easily seen that a higher number of photons $n = \hat{a}^\dagger\hat{a}$ in the resonator results in a larger shift of the qubit frequency. As this effect depends on the photon number in the resonator, it is known as *number splitting*. For a dissipative system this effect becomes continuous, and is described as *AC Stark shift*. When doing continuous spectroscopy it is therefore advised to use low resonator drive power to populate the resonator with only a few photons. Alternatively, effects related to the AC Stark shift can be avoided by doing *pulsed spectroscopy*. Pulsed spectroscopy ensures that the resonator is not populated with photons while the qubit is driven. For that in *SweepSpot frontend* the microwave generators that drive the readout resonator and the qubit have to have the "Modulation" option enabled. Furthermore a pattern has to be loaded to the AWG that performs pulsed spectroscopy. Such an example was shown in Figure 3.4. There the resonator drive is only switched on during the time the FPGA records data. Before, the qubit is excited. At the moment the resonator is populated with photons, the qubit drive is turned off. A comparison of the two spectroscopy types is depicted in Figure 6.7.
As the qubit is not driven anymore during recording time, it starts to decay. The duration the FPGA records data should therefore be on the same order as the qubit coherence time to not measure a lot of noise data after the qubit has decayed.

6.3.4 Checking the behavior of the qubit

Now it needs to be checked whether the detected resonance actually belongs to a qubit transition and whether it is the correct qubit. For the following experiments the uncertainty on the qubit frequency needs to be small compared to the bandwidth of the qubit excitation pulses. There are several ways to test whether a resonance originates from a qubit transition.
As a first indicator, it can be checked if the $|g\rangle \leftrightarrow |f\rangle$ /2 transition below the presumed $|g\rangle \leftrightarrow |e\rangle$ resonance can be seen. However, not seeing a second transition in this region does not exclude the presumed $|g\rangle \leftrightarrow |e\rangle$ resonance to origin from a qubit as the fabricated qubit could have a different anharmonicity than designed.

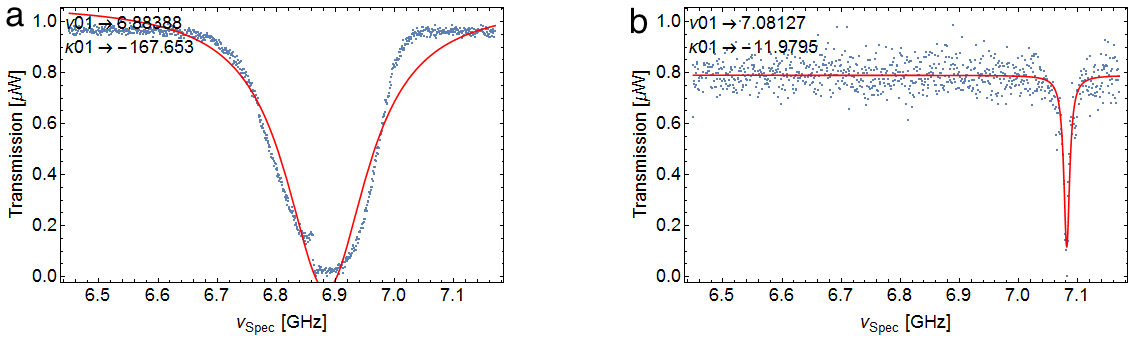


Figure 6.7: (a) Unpulsed qubit spectroscopy with high qubit drive power. There, the actual qubit frequency is shifted by the AC stark shift due to interactions with the photons in the resonator. (b) Pulsed spectroscopy of the same qubit with low power. Now the real transition frequency of the qubit is revealed as the AC Stark shift got suppressed.

A more reliable test is whether the resonance can be shifted by applying a magnetic flux bias. This test can be performed using *SweepSpot frontend* with a DC Voltage sweep of one of the coils. If the resonance moves it is likely to originate from a qubit. However, also this test can fail in some cases even though the resonance belongs to a qubit transition. In particular, the qubit could work well but be not tunable via an external magnetic field. Either the qubit design could cause this (e.g. a screening current around the qubit shielding it from external magnetic fields when using coils for biasing the magnetic field) or an unintentional problem during fabrication. However, the qubit test sample from Section 6.1 passed this test, as shown in Figure 6.8.

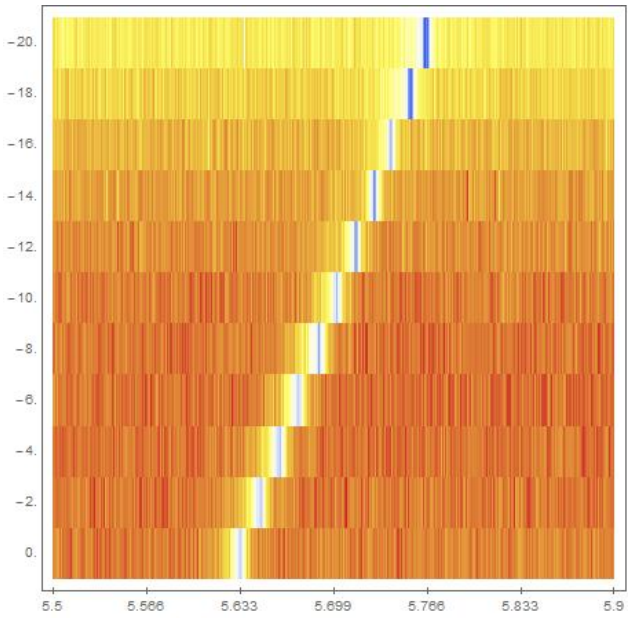


Figure 6.8: The dependency of the qubit 4 frequency on the flux bias.

Nonetheless, the most reliable test of whether a resonance belongs to a qubit is trying to drive

Rabi oscillations with the resonance. Rabi oscillations are a unique feature of a qubit. If they are observed it can be concluded that the resonance belongs to a qubit. The experiment works as follows:

1. The qubit is prepared in its ground state, represented by a Bloch vector pointing towards the negative direction of the z-axis on the Bloch sphere.
2. A pulse with amplitude A and length τ is applied to the qubit. In the Bloch sphere picture this operation can be represented by a rotation σ_x around the x-axis. The rotation angle is proportional to the pulse amplitude A, its length τ and the drive rate.
3. The qubit population is extracted using a projective measurement.
4. The procedure is repeated either with amplitude $A + \delta$ and length τ or with amplitude A and length $\tau + \delta$.

In each step either the amplitude of the pulse or the pulse duration is varied. Recalling the interaction Hamiltonian of the Transmon, Eq. 2.23, one can derive the measured population when the qubit is driven at its $|g\rangle \leftrightarrow |e\rangle$ transition frequency (ignoring damping) as [3]

$$|p_e|^2 = \sin^2(\Omega_R \tau/2) \tag{6.4}$$

where Ω_R denotes the Rabi frequency. The population therefore oscillates in time, known as *Rabi oscillations*. The result is depicted in Figure 6.9. From a sinusoidal fit the maximum amplitude can be extracted, which is expected to correspond to $|p_e| = 1$. The corresponding Rabi amplitude Ω_R corresponds to a π amplitude on the Bloch sphere.

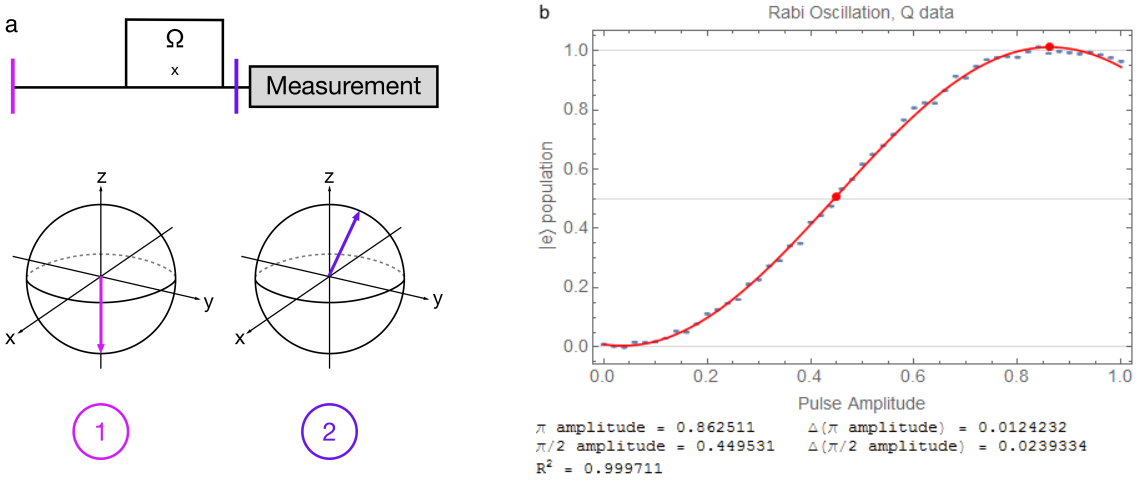


Figure 6.9: (a) Schematic representation of a Rabi experiment. The vertical line on the top represents the pulse scheme of the experiment. The two Bloch spheres show the qubit state at the times marked on the pulse scheme. First, the π pulse rotates the Bloch vector around the x-axis with angle A. This is followed by a projective measurement of its z-component. (b) A Rabi experiment with the bare experiment data (blue), a sinusoidal fit (red line) and the extracted π and $\pi/2$ pulse amplitudes (red points).

Finally, one may want to check if the resonance actually belongs to the expected qubit on the sample. In such a sample design as in Figure 6.1 it was discovered that it is possible to drive qubits by charge lines of neighboring qubits. Therefore, it can be checked whether one looks at the expected qubit by comparing the qubit spectroscopy responses when driving the qubit

through different charge lines. Figure 6.10 shows that the largest response is measured when the qubit is driven with charge line 2, inferring that charge line 2 is the one that directly couples to the observed qubit and we therefore look at qubit 2. However, the qubit could also be excited using the drive line of qubit 3 or 4. It is not fully understood which effect caused the crosstalk for this sample. One possibility is that the Purcell filter which connects all readout resonators and like this all qubits served as a coupling bus. However, as the resonators work as bandpass filters, crosstalk signals caused by the Purcell filter are expected to be lower.

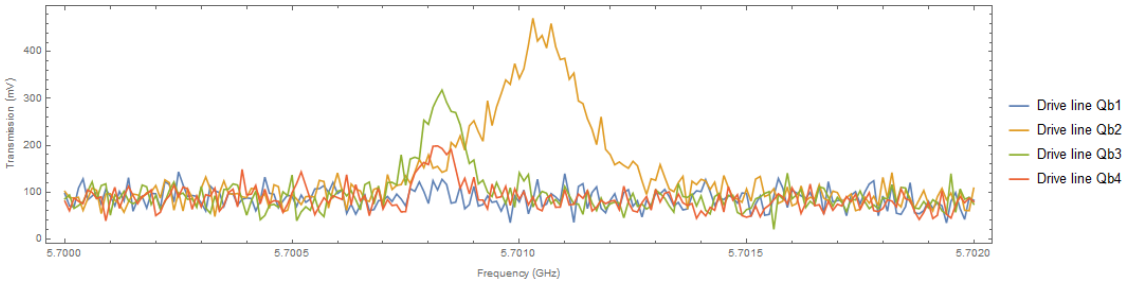


Figure 6.10: Spectroscopy of qubit 2 driven through charge lines of different qubits. The biggest crosstalk was observed for charge line 3, followed by line 4. As expected, qubit 2 could not be tuned through the charge line of qubit 1 as its readout resonator was shortened.

6.3.5 Anharmonicity

The next step is to measure the anharmonicity of the qubit. Taking into account only the first three energy levels, the anharmonicity is defined as

$$\alpha = \omega_{ef} - \omega_{ge} \tag{6.5}$$

where the $|g\rangle \leftrightarrow |e\rangle$ transition frequency is measured at low drive power to suppress effects from the AC Stark shift. The $|e\rangle \leftrightarrow |f\rangle$ transition can be calculated by measuring the $|g\rangle \leftrightarrow |f\rangle /2$ transition at high power and using the relation $\omega_{ef} = 2\omega_{gf/2} - \omega_{ge}$. An example of a $|g\rangle \leftrightarrow |f\rangle /2$ transition is shown in Figure 6.6. The anharmonicity allows to calculate E_C and E_J using the function *FindEcEj* from the Qudev Mathematica library. This function extracts E_C and E_J from a fit of the Mathieu equations solutions of the Transmon Hamiltonian using the measured values of ω_{ge} and $\omega_{gf/2}$.

6.3.6 Exact Transition Frequency

The exact transition frequency of a qubit can be found with a Ramsey experiment. As illustrated in Figure 6.12 it works as follows [41]:

1. The qubit is prepared in its ground state. This corresponds to the Bloch vector pointing towards the -z direction.
2. A $X(\pi/2)$ pulse is applied to the qubit, causing the Bloch vector of the qubit state to rotate around the x-axis by 90°.
3. The Bloch vector of the qubit freely evolves around the z axis with frequency $\Delta\omega$ which is the detuning of the drive from the qubit transition frequency.

4. A $Y(\pi/2)$ pulse is applied to the qubit. Like that the x component of the state vector is projected onto the z axis. The y component remains the same. This allows to determine the sign of $\Delta\omega$.
5. A projective measurement is performed. In the Bloch sphere picture we actually measure the z component of the state vector. More precisely, the measured population follows the distribution [3]

$$p_e = \frac{1}{2} + \frac{1}{2} e^{-\tau/T_2^*} \cos(\Delta\omega\tau) \tag{6.6}$$

where T_2^* is the dephasing rate of the qubit ensemble. The cosine term represents the Rabi oscillations with frequency $\Delta\omega$. The exponential term describes the decoherence of the qubit state. It can be directly seen from Eq. 6.6 that for $\tau >> T_2^*$ the qubit reaches the maximally mixed state $\rho = \frac{1}{2}(|0\rangle\langle 0| + |1\rangle\langle 1|)$.

Then the experiment is repeated for different separation durations τ, causing different dephasings and therefore different measured populations. The result are Ramsey fringes, as depicted in Figure 6.11.

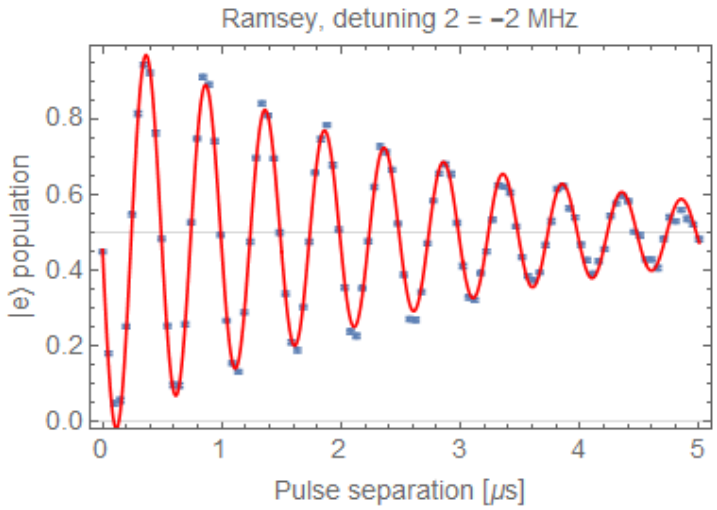


Figure 6.11: Ramsey fringes observed with a Ramsey experiment (blue points). The red line represents the according model, a sinusoidal, decaying function, with parameters optimized using a least-squares method.

From the Ramsey fringes the detuning of the drive to the qubit transition frequency can be extracted which leads to an exact value of the qubit frequency. Of course this requires to perform the experiment with a drive frequency that is already nearly on-resonant with the qubit.

Additionally to the exact qubit frequency this experiment allows to extract the dephasing time T_2^*. It differs from T_2 as T_2^* is the result of an ensemble average [14]. As a Ramsey experiment is repeated several thousand times, small fluctuations in the qubit frequency (and therefore in $\Delta\omega$) slightly change the detuning in step 3 and therefore lower the net measured coherence time. Therefore, we always have $T_2^* < T_2$. To measure the natural T_2 coherence time of a qubit, a spin-echo experiment can be performed, as explained in Section 6.3.8.

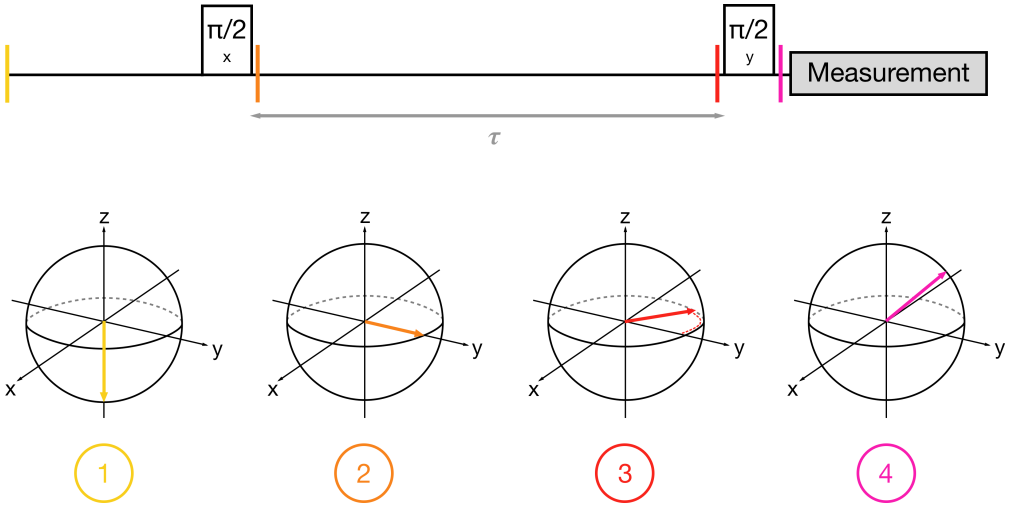


Figure 6.12: Schematic representation of a Ramsey experiment. Two $X(\pi/2)$ pulses are applied to the qubit, separated by τ. Then, directly after the second pulse the qubit state is measured.

6.3.7 QScale calibration

In general, a pulse applied on a qubit that should drive the $|g\rangle \leftrightarrow |e\rangle$ transition also drives the $|e\rangle \leftrightarrow |f\rangle$ transition to a certain amount. The latter is mostly unwanted and needs to be suppressed. For that reason so-called DRAG (Derivative Removal of Adiabatic Gate) pulses are used. As Figure 6.13 illustrates, a DRAG pulse on ω_{ge} minimizes its effect on the $|e\rangle \leftrightarrow |f\rangle$ transition whose frequency is close. At the same time, the pulse contribution on the $|g\rangle \leftrightarrow |e\rangle$ transition frequency is still large. The amplitude of the DRAG pulse is given by [3] $A(t) = \sqrt{\varepsilon_x(t)^2 + \varepsilon_y(t)^2}$ and its phase is described by $\varphi = \arctan(\varepsilon_y(t)/\varepsilon_x(t))$. $\varepsilon_x(t)$ is chosen as a Gaussian [13], truncated at 3σ around its maximum and starting and ending with zero amplitude. $\varepsilon_y(t)$ is described by $\varepsilon_y(t) = -\frac{q_s}{\alpha}\frac{d}{dt}\varepsilon_x(t)$ where α is the anharmonicity and q_s is called the QScale factor. The shape of the DRAG pulse needs to be optimized, i.e. the pulse contribution at the $|e\rangle \leftrightarrow |f\rangle$ transition frequency, to be minimized by calibrating the QScale factor.

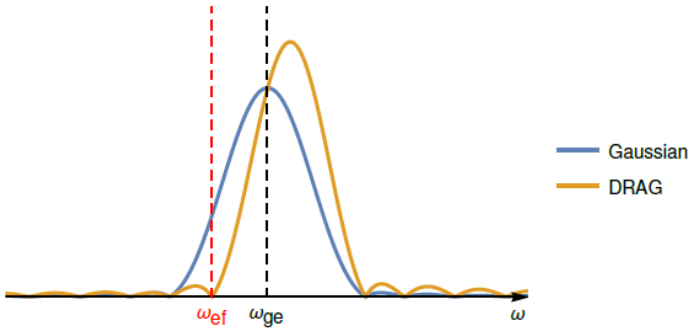


Figure 6.13: A DRAG pulse driving the $|g\rangle \leftrightarrow |e\rangle$ transition and minimizing its effect on the $|e\rangle \leftrightarrow |f\rangle$ transition whose frequency is close. Figure adapted from [13].

During calibration, q_s is swept in a particular range and for each intermediate value 3 gates are applied: $X(\pi/2)X(\pi)$, $X(\pi/2)Y(\pi)$ and $X(\pi/2)Y(-\pi/2)$. All those rotations should theoretically result in a qubit population of $|p_e| = \frac{1}{2}$. However, as Figure 6.14 shows, the qubit population strongly varies with the scale factor q_s. After the experiment the ε_y amplitude is therefore scaled with the q_s value that leads to the best results (i.e. to the qubit populations closest to 0.5).

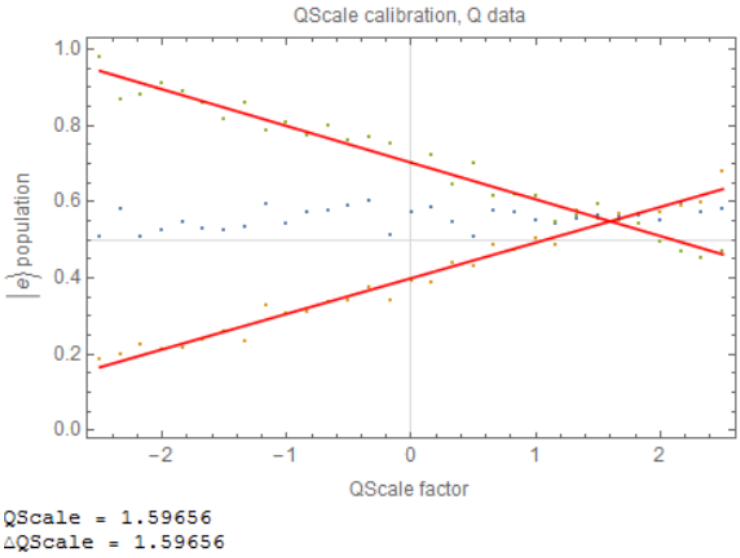


Figure 6.14: The result of a QScale experiment. The three colors represent the different applied gates $X(\pi/2)X(\pi)$, $X(\pi/2)Y(\pi)$ and $X(\pi/2)Y(-\pi/2)$. Fitting the results of each gate with a linear function and calculating the crosspoint of the functions gives the optimal QScale factor.

6.3.8 Coherence Times

Now we can measure the energy relaxation time T1 which was introduced in Section 2.6. This is done by applying a $X(\pi)$ pulse to the qubit, as illustrated in Figure 6.15 (a), and then measuring its state as a function of time. An exponential decay as shown in Figure 6.15 (b) can be observed, from which, using

$$|p_e| \propto e^{-t/T_1}, \tag{6.7}$$

T_1 can be extracted.
The dephasing time T_2 can be found with a spin-echo experiment, developed for NMR technology [25] but directly applicable to superconducting qubits as well. A spin-echo experiment works similarly as a Ramsey oscillation experiment (see Figure 6.16):

1. The qubit is prepared in its ground state
2. A $X(\pi/2)$ pulse is applied to the qubit. Like this the Bloch vector rotates around the x-axis by a 90° angle.
3. The Bloch vector freely evolves for a duration $\tau/2$ around the z-axis with frequency $\Delta\omega$ (see Ramsey experiment).
4. A $Y(\pi)$ pulse is applied to rotate the Bloch vector around the y-axis. This accounts for effects that change the qubit frequency on a longer time scale than the duration of one

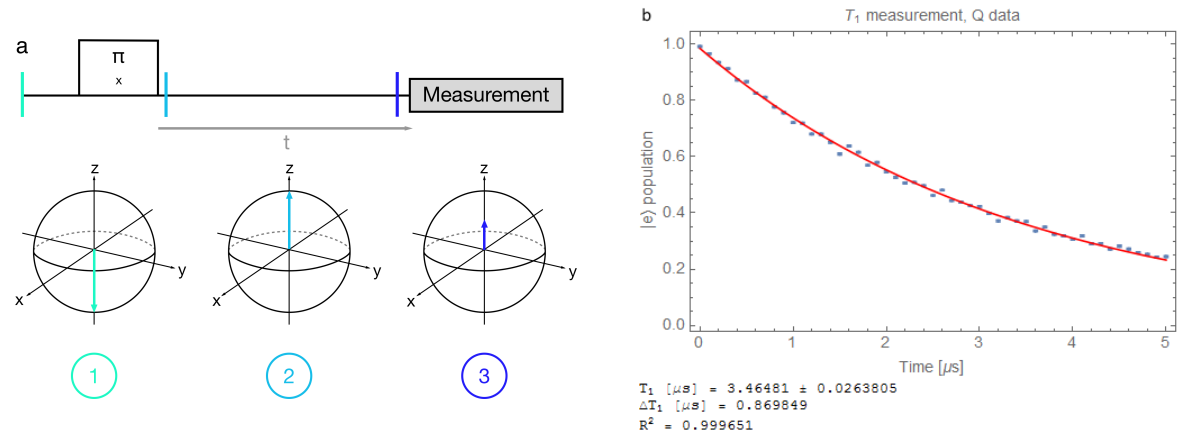


Figure 6.15: a) Schematic representation of a T1 measurement. A $Y(\pi)$ pulse is applied to the qubit and its population is measured as a function of decay time. b) Observation of the energy relaxation of a qubit with a T_1 experiment.

experiment. The Bloch vector then freely evolves around the z-axis, again reaching the y-axis.

5. After the same free evolution time $\tau/2$ as in step 3 the Bloch vector is again rotated around the x-axis by the angle $\pi/2$.
6. A measurement is performed. Again, we effectively measure the z-component of the Bloch vector. The length of the z-component got modified due to the decoherence of the qubit state.

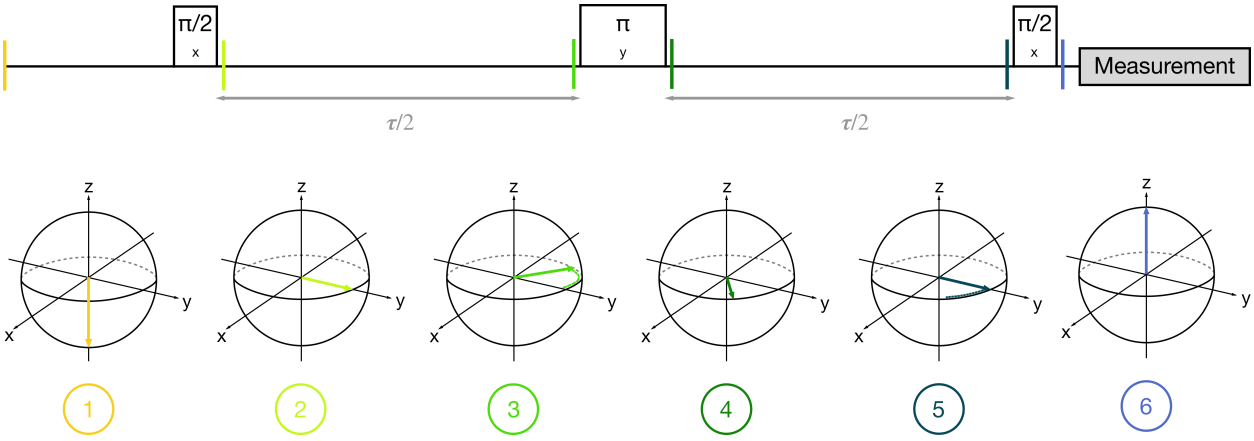


Figure 6.16: Schematic representation of a T_2 experiment. Additionally to the sequence used for a Ramsey experiment, a $Y(\pi)$ gate is applied in the middle of the sequence to cancel effects due to changes in the qubit frequency.

6.4 Tracking the qubit

So far we have characterized the qubit for a reference external magnetic field (e.g. all coils tuned to 0V). Now we want to see how the properties of the qubit change when a flux bias is applied. This process is called *tracking the qubit*. A tracking experiment cannot easily be performed with *SweepSpot frontend* as sweeping the magnetic field also changes the resonator frequency (see Section 2.4). At some magnetic field configuration the resonator is not driven at its resonant frequency anymore and the transmitted signal is only weakly dependent on the qubit state. This is the main motivation a tracking sequence is needed to measure the characteristics of the qubit at various magnetic field configurations. To track a qubit, the readout frequency has to be adjusted in each step. In general, a Tracker sequence performs the following steps:

1. The tracker software sets the coil voltages to a predefined configuration.
2. A resonator spectroscopy determines the optimal readout frequency. The software then automatically uses this frequency for the following experiments with that magnetic field configuration.
3. A qubit spectroscopy experiment determines the approximate qubit frequency. The software automatically uses this frequency, added by the IF frequency (see Section 3.3) to perform the following experiments via a mixer.
4. A sequence of calibration experiments is performed, using corresponding AWG patterns. Example: Rabi, Ramsey, Rabi, QScale, Rabi, T1, T2.

Then this sequence repeats for the next magnetic field configuration.

As part of this project, a tracker functionality was added to *QubitCalib* (see Section 5.6). Like this the qubit can be tracked as shown in Figure 6.17. Figure 6.17 (d) shows that the software actually tracks the qubit such that one does not need to scan the whole frequency range in each step. At the moment the function assumes no change of the magnetic field configuration as it centers the next frequency sweep around the previous frequency. Figure 6.17 (a) shows that also the readout resonator moves with the magnetic field, even though the dependency is not as strong as in the case of the qubit. Figure 6.17 (b) shows the magnetic flux dependency of the qubit and in part (c) the relation to the coherence times can be seen.

A tracker experiment then allows to determine relevant parameters of the qubits, especially $E_{J,max}$. This can be done by fitting the qubit frequency-to-flux bias ratio (see Figure 6.17 (b)) to an according model. As the fit model the Transmon solution of the Cooper Pair Box Hamiltonian is used (Mathieu functions, see Section 2.2), plugged in to a Hamiltonian that describes a qubit with with a coupled readout resonator. The fitting parameters are the bare resonator frequency ω_r, the resonator-qubit coupling strength g, the maximal qubit frequency $\omega_{q,max}$, the maximal Josephson energy $E_{J,max}$ and the flux quantum Φ_0. This allows, when having a good estimate for some of the parameters, to extract the others.

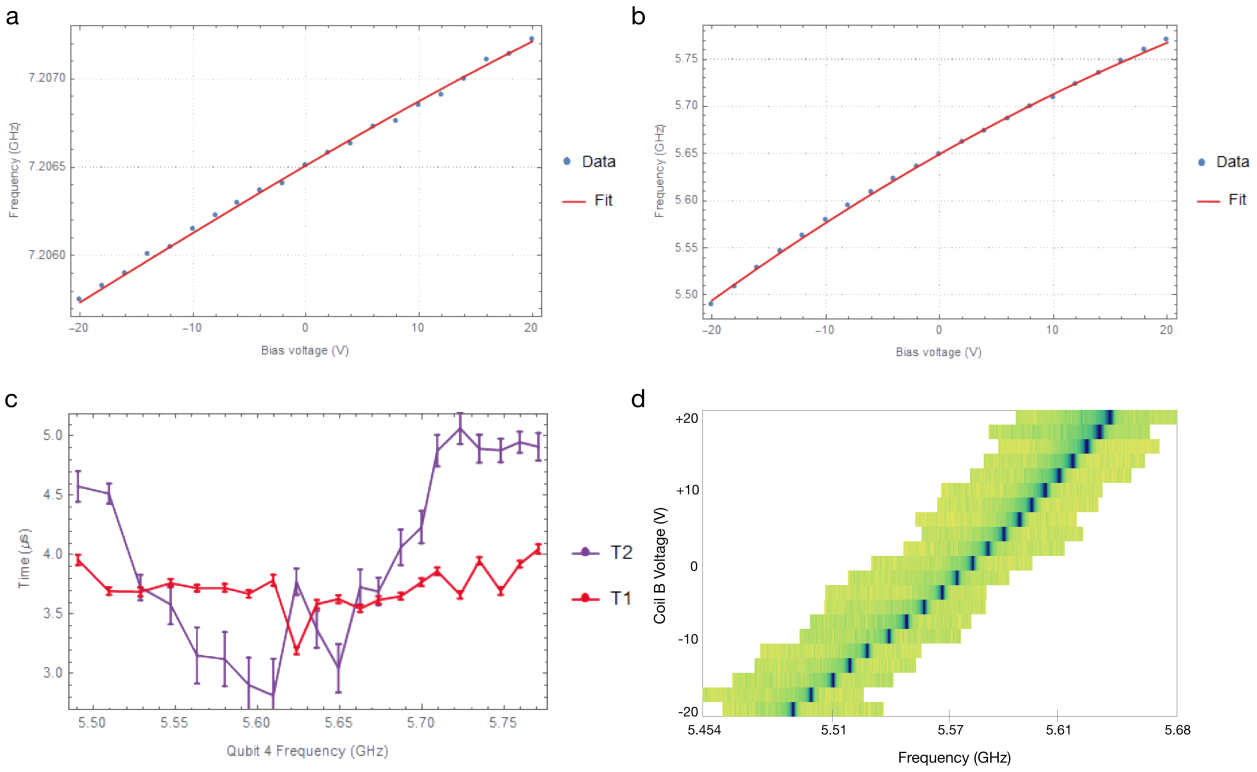


Figure 6.17: Data gathered from tracking qubit 4 by sweeping coil B underneath the sample. (a) The resonator 4 frequency, moving with the biased magnetic flux. (b) The qubit 4 frequency and its magnetic field dependency. (c) Extracted T_1 and T_2 times at the different magnetic field settings, respectively the different qubit frequencies. (d) All the data gathered with the qubit spectroscopy experiments. This Figure shows that the software actually tracks the qubit, i.e. it estimates the expected frequency of the next step instead of sweeping over the full range each time.

6.5 Further experiments

There are more experiments that can be performed to characterize a sample. A few examples are:

- Measurement of the exact value of the dispersive shift of a resonator by comparing its spectrum in the ground- and excited state
- Observation of the anti-crossing of two qubits for a sample with coupling resonators
- Flux line cross talk analysis for a sample with flux lines
- Randomized benchmarking to estimate the fidelity of single- and multi-qubit gates

After the characterization of a sample one should be able to determine whether the chip is suitable for a particular experiment. Furthermore, particular steps of the characterization may have to be repeated at later stages. For example, qubit spectroscopy is a common experiment that is regularly repeated for diverse reasons, even though the sample is already well characterized.

Chapter 7

Conclusion

7.1 Summary

In this project a new LabView software suit for performing experiments in the Qudev laboratory was developed. *SweepSpot* was implemented with a clean and well-structured concept. It comes with built-in implementations of new devices, it is based on an optimized code which allows faster experiments and it provides new features such as multidimensional sweeps. Additionally the user interface was improved with the aim to generate a good workflow. The user interface provides detailed status feedback and an experimenter is able to export data directly to Mathematica using it. The *SweepSpot* framework was designed such that it can be easily extended for future additions. In particular, for the first time spectroscopy experiments were automized and their modules were added to the calibration automation LabView software *QubitCalib*. The individual modules can also be used for other implementations. A more convenient way to track qubits was developed to perform the important magnetic flux sweeps with ease. Thanks to detailed documentation of the software in this thesis and on the Qudev Wiki website, members of the laboratory can use the LabView VIs and Mathematica functions to perform their experiments with more advanced tools or to automize certain parts of their measurements. Furthermore, this thesis suggests a systematic approach to characterize a sample. It provides an overview of the required steps to find the relevant parameters of a chip and it shows how the new software suit can be used to perform these experiments.

7.2 Proposals

As software development is a never-ending process there are several additions to the SweepSpot framework one could think of.
First, it would be beneficial to have an option to save data throughout the experiment. This would prevent data loss in case of an error at a late stage in the experiment.
It could also be investigated whether the data format (.ini) of the configuration files should be changed for faster loading/writing times and a better handling of the content of the files. Interesting candidates would be to format the files in JSON or XML.
As there will be more DC sources connected to the setup in the future one needs to update the DC source instrument. It should be able to handle DC sources of different devices.
Then, as explained in Section 4.3, we found a LabView VI connected to the use of an FPGA that

was responsible for a big fraction of the duration of an experiment (*"FPGA Setup.vi"*). I strongly suggest to check whether there is a bug in the FPGA implementation that slows down the experiment.
Also there are a few instruments that are not yet implemented to SweepSpot, namely the UHFLI acquisition device of *Zurich Instruments* and the DAQ of *Acquiris*.
Another feature that was regularly asked about is a mechanism that allows to read certain parameters of devices (e.g. the current coil voltages), to collect and store them in a configuration file for future analysis. On a higher level the tracker functionality in QubitCalib can be optimized by covering other measurements than linear magnetic field sweeps or by automatically adjusting the drive power during the spectroscopy experiments. "Tracker" could even be built to QubitCalib as a subroutine itself which would allow to get rid of a long list of tracker step operations on the QubitCalib frontend.
Furthermore, a plotting VI that could be accessed by various VIs would be useful. During an experiment, such a VI would show the data gathered so far and it would allow to analyse the data on the fly, i.e. by looking at different slices of it or making histograms out of it.
Finally, one can also think of more advanced automation routines. For example, a VI could be built that would park the qubits iteratively, based on a Nelder-Mead algorithm. Like this one would not need to generate a flux model of the sample. However, it should first be calculated how long such an algorithm would need to execute and whether it could succeed in a reasonable amount of time.

Acknowledgments

First, I would like to thank Prof. Andreas Wallraff a lot for giving me the opportunity to conduct my Master's thesis in his group. It was an interesting project in a highly fascinating field and I enjoyed it a lot.
Furthermore, I am very grateful to my main advisor Johannes Heinsoo for many valuable discussions and his guidance during this project. I gained a lot from his knowledge and from his great programming skills and thanks to our jogging activities he even managed to keep me in shape during these six months.
Moreover, I want to thank Yves Salathé for practical instructions in the laboratory and for being supportive whenever I had questions. I also owe thanks to Samuel Haberthür who made the first steps in the implementation of SweepSpot before I joined the team. In addition, Fadri Grünenfelder developed some of the Mathematica spectroscopy fitting functions as his semester project.
Furthermore, I am grateful to all Qudev members for creating a friendly and enjoyable atmosphere in the laboratory. And finally, I want to thank my girlfriend and my family for their great support throughout my studies, especially during this Master's project.

Appendix A

Measurement configurations of time statistics experiments

The following table shows the relevant experiment settings for the data shown in Section 4.3. The number of spots was changed as shown in Figure 4.23.

Table A.1: SweepSpot speed statistics, shown in Figure 4.23.

Soft avgs	Settl. Time	Rep. rate	dFactor	Samples	Segments	Averages	No. of MWGs
1	0 s	10 μs	2	250	1	1k	3

Appendix B

Adding features to SweepSpot

SweepSpot is built such that adding new features can be implemented with a clear procedure, opposed to Cleansweep where the implementation of new methods and instruments was cumbersome. To stick to the concept of the software and to keep the code clean one should follow some general rules. These will be presented in the following paragraph. Afterwards it will be shown how to implement new instruments or sweep types and how to use SweepSpot as a subVI in a higher level function.

B.1 General remarks

One of the main motivation to build SweepSpot was that the untidy implementation of Cleansweep limited the development of new features. Also the code was often unclear. This fate should not happen to SweepSpot. Such that new users do not brake with the concept of SweepSpot they should follow the 3 golden rules of SweepSpot implementations:

1. **Keep the concept**
   This sounds obvious but actually it is the most important rule: make sure you understand the basic concept of SweepSpot if you implement new features. Do not try to reinvent VIs that are already here and implement the additions at the correct place.
2. **Choose the level of the implementation wisely**
   This rule directly follows after the first one. Whenever you implement a new feature to the SweepSpot framework, make sure that you do it at a sensible level. As a rule of thumb, *SweepSpot main* and its subfunctions should be kept as simple as possible. Changes in these VIs should be either of fundamental nature or they should provide general new functionality such as the addition of a new instrument or a new sweep type. Features that involve different stages of an experiment or repetitions of certain steps should be implemented as higher-level functions, calling *SweepSpot main* whenever needed, e.g. in a loop. Like this one ensures that the ground structure of the framework (*SweepSpot main*) is kept simple and clean whereas the more complicated functions are on a higher level (see Section B.5).
3. **KISS: keep it stupid simple**
   This is a general rule for software development and should also be in mind for modifications to SweepSpot. The simpler an implementation, the better. Clean and easy implementations of features make it easy for other people to understand the code and to build upon

it or to reuse it. If one has to write a complicated piece of code it is very helpful to add comments to the LabView block diagrams that explain what is going on in which step.

Furthermore, one should make sure that new VIs carry reasonable names and paths and an icon that explains the basic purpose of the function at first sight. If everyone respects these 3 golden rules of SweepSpot implementations it probably can be ensured that there will not have to be a software cleanup for a long time.

B.2 Adding a new sweep type

This paragraph describes how to add a new sweep type such as ”Frequency” or ”MW Power”.

SweepSpot frontend

1. On the front panel of *Sweep Spot frontend* there is the list of sweep dimensions. To add a new sweep type to choose from this list one has to right-click one of the list elements (which is a LabView type definition called *Sweep dimension element*) and add the new type. For simplicity, for now it will be called ”Magic type”.
2. Still on the frontend, in the *Sweep controls* Section one has to add a tab for ”Magic Sweep”. Optimally one creates a type definition that contains all the controls of this sweep type.
3. On the block diagram of *SweepSpot frontend* there is the new data cluster of the ”Magic sweep” control that has just been added. It now should be combined with the other sweep types inside *Create Sweep Definitions.vi* to add this type to the *Sweep Definitions array*.
4. Now the new sweep type should be added to the functions that write and load the frontend configuration files, namely *Unbundle SweepSpot settings.vi* and *Save SweepSpot Frontend data.vi*. These can be found on the frontend in the consumer main case structure under the cases ”Load” and ”Save”.
5. It is important that the new sweep types are also added to the type definition *SweepSpot settings.ctl*. In particular, inside *SweepSpot settings.ctl* the ”Magic sweep” type has to be added to the ”Sweep Definitions” subcluster.
6. Finally, to populate the controls on the frontend one has to add a reference to the new sweep type to the *Sweep Controls Reference* type definition (found in the case ”Load” on the frontend Block Diagram).

SweepSpot generator

In this VI the *Spot Array* is built. As explained in Section 4.1.3, this is a list of spots containing the relevant parameters of the instruments that have to be set in each spot. In *Spot Array element generator* there has to be added a new function, i.e. a new case, that creates a *Spot array* with the parameters to be set in each spot. Additionally, there should be output relevant metadata about the new sweep type, i.e. start, stop and step size of the range of the parameter that is swept and the total number of steps. Basically one should add here all information that one later wants to have written to the header of the data files. In certain special cases one may even need to change the type definition *Sweep info* for that, but normally one can use the ”Start”, ”Stop”, and ”Step size” parameters that are already there. Every time one updates *Sweep info* one should also

update *Feedback controls.ctl*. This type control contains a reference to *Sweep info*, so one just has to replace the current reference with a new one, carrying the name "Sweep info".

Save measurement data

Finally, the defining information about the new sweep type has to be added to the header of the data files. In the SubVI *MakeConfigStringForSweepSpot→Add Sweep Info to Header* one has to add a new case that writes the relevant information of the new sweep type to the header. As the header is formatted as a Mathematica association one should make sure that this is also the case for the new sweep type. Now the new sweep type is officially implemented.

B.3 Adding a new instrument

This paragraph describes how to add a new instrument to the SweepSpot framework. For all devices that *measure* the steps of the next Section B.4 should be followed.

SweepSpot frontend

1. On the front panel of *SweepSpot frontend* there has to be added a new tab to the "Instrument settings" section. There the new controls can be placed. All new controls should be bundled to a LabView type definition.
2. On the block diagram of *SweepSpot frontend* the cluster with the settings of the new device has to be wired to *Create Instrument Settings.vi* where the new controls can be added to the *Instrument settings* variant.
3. To provide frontend settings loading/saving functionality one has to add the property node of the new control to the "Load" case on the block diagram. Then the new instrument has to be added to the two functions that write and read the frontend configuration file. For that follow steps 4 and 5 of Section B.2 → SweepSpot frontend.

SweepSpot Generator does not have to be changed.

SweepSpot.vi

Now the functions that call the devices to change certain parameters have to be added.

1. In *Prepare Sweep → Initialize Instrument* the string constant the array loops over has to be completed with an entry of the new instrument. Then there has to be added a new case in the case structure containing the VI that initializes the device. If no initialization is needed, this step can be skipped.
2. In *Prepare sweep → Base set* a new case has to be added where the default parameters of the instrument that one sets before the first spot are chosen. One can get an idea by looking at the corresponding implementations of the other instruments.
3. In *Prepare sweep → Set instrument* the function that actually sets new values of device-specific parameters has to be added. For that, analogously to *Initialize Instrument*, one has to complete the string array the VI loops over and one has to add a new case in the loop. There one places the VI that sets the new parameters. Note that the VI *Set instrument* is called in every spot (see Section 4.1.4).

4. Finally, one should modify *Motion reversal* similarly to the other two VIs *Initialize instrument* and *Set instrument*. In the new case of the case structure there has to be added the function that "cleans up" the device, i.e. that makes it ready for the next experiment. Usually this involves closing open VISA handles.

Save measurement data does not have to be changed.

B.4 Adding a new acquisition device

This section describes how to implement a device that performs a readout.

SweepSpot frontend

First one needs to add frontend controls to *SweepSpot frontend*.

1. On the front panel of *SweepSpot frontend* there has to be added another tab to the section called "Acquisition Instrument settings". It makes sense to have nested type definitions as with the FPGA. Overall there should be a single type definition which may contain the sub-type definitions. Like this one can use this single type definition of the acquisition device (e.g. "FPGA.ctl") throughout the whole software. The device should also have a main "Enabled" button to activate/deactivate the readout with this device.
2. On the block diagram of the frontend one then needs to append the new device settings to *Instrument settings*.
3. Additionally the controls of the new device should be added to the functions that write/load the frontend configuration files. For that one has to follow the steps 4 and 5 of Section B.2.

SweepSpot generator

Here one needs to add information about the new device to the *Sweep Info* cluster.

1. In the subVI *Sweep Info generator* there has to be added a new case for the new readout device that writes the name of the new device. This ensures that if the new device is enabled the name of the device will be added to the header in the data files.
2. In *Sweep Info generator→Dimension calculator* the size of the lowest dimension of the measured data has to be calculated. The 2nd lowest data dimension is always 1D (see Section 4.1.4). For FPGA firmwares *TVMode* and *Correlator* this represents a time trace. For an FPGA therefore this VI calculates the length of this 1D array, i.e. the number of points in the time trace. There should be a similar case for the new acquisition device.

SweepSpot.vi

Here where we actually perform the measurement one needs to add functions that gather the data read out by the new device.

1. First the new acquisition device has to be initialized. For that, similarly to the equivalent step in Section B.3, one has to implement the corresponding function in *Prepare sweep* → *Initialize instrument*.
2. In *Spot.vi* → *Make acquisition ready* one has to tell the acquisition device to be ready for the next measurement using a subVI that accesses the new device.

3. The readout is performed in *Spot.vi* → *Readout acquisition*. There the subfunction that performs the readout has to be added to the For-loop and the case structure by adding a corresponding new case for the new instrument. The measurement data for each spot has to be 1D. It therefore has to be ensured that higher dimensional data is flattened. The unflattening is done automatically in *Save measurement data*.
4. Finally, in *Motion reversal* the software part of the readout device has to be "cleaned up" and all open Visa handles should be closed.

Save measurement data

Here one only need to set a unique infix in the filename which indicates that the dataset in this file was acquired by the new readout device. This can be done in *Save measurement data* → *Save Instrument Data* → *Filename Creator*.

B.5 Using SweepSpot as a module

One of the big advantages of SweepSpot compared to Cleansweep is that *SweepSpot main* can be easily used as a module in a higher-level VI. The SweepSpot framework was designed such that basically every VI can be used as a module, especially the most important VI *SweepSpot main*. This section shows how to embed it into another LabView function.
The only thing that has to be taken care of if a LabView VI calls *SweepSpot main* is to wire the proper datatypes to it. There are required and optional inputs as the following overview shows. More information about those data types can be found in Section 4.1.2.

- **Save info**
  This type definition contains
  - the filenumber;
  - the filename in combination with its path;
  - a boolean "Save" indicating whether any data files should be written;
  - the path to a pattern configuration file if one wants to save a pulse configuration file (optional);
  - a ring element/integer indicating the file format (ASCII or binary);
  - a suffix to the filename if needed.
- **Sweep Dimensions**
  A list of ring elements, where each one should be an element of the "Sweep Dimensions element" type definition, that specifies which sweep types should be used and in which order. The first array element corresponds to the fastest dimension.
- **Sweep Instruments**
  A variant containing elements that correspond to the sweep types one wants to use (e.g. "Frequency 1", "DC Voltage"). It is required to add an element for every different sweep type one has specified in *Sweep Dimensions*.
- **Instrument settings**
  A variant containing as elements all information about the relevant devices connected to the setup.
- **Soft averages (Optional)**
  The number of times the software should repeat the experiment and average over the re-

sults.

- **Error (Optional)**
  The standard LabView error wire input.
- **Feedback (Optional)**
  A wire of type *Feedback.ctl* containing references to higher-level elements for plotting and status updates such as a status box or a progress bar.

As *SweepSpot main* is designed as a sequentially reusable module there are also wires that can be used as outputs. In particular, this is the standard error output and the (potentially modified) *Instrument settings*. The concept is that *Instrument settings* is the wire that flows through the whole higher-level VI. It may be modified by some subVI (e.g. *SweepSpot main*) and then reused by another subfunction.

Bibliography

[1] Anatole Abragam. *The principles of nuclear magnetism*. Number 32. Oxford university press, 1961.

[2] NH Balshaw. Practical cryogenics, 2001.

[3] Matthias Baur. *Realizing quantum gates and algorithms with three superconducting qubits*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 20359, 2012, 2012.

[4] Charles H Bennett. Quantum cryptography: Public key distribution and coin tossing. In *International Conference on Computer System and Signal Processing, IEEE, 1984*, pages 175–179, 1984.

[5] Charles H Bennett, Gilles Brassard, Claude Crépeau, Richard Jozsa, Asher Peres, and William K Wootters. Teleporting an unknown quantum state via dual classical and einstein-podolsky-rosen channels. *Physical review letters*, 70(13):1895, 1993.

[6] Simon Berger. *Geometric phases and noise in circuit QED*. PhD thesis, Diss., Eidgenössische Technische Hochschule ETH Zürich, Nr. 22524, 2015, 2015.

[7] Alexandre Blais, Jay Gambetta, A Wallraff, DI Schuster, SM Girvin, MH Devoret, and RJ Schoelkopf. Quantum-information processing with circuit quantum electrodynamics. *Physical Review A*, 75(3):032329, 2007.

[8] Alexandre Blais, Ren-Shou Huang, Andreas Wallraff, SM Girvin, and R Jun Schoelkopf. Cavity quantum electrodynamics for superconducting electrical circuits: An architecture for quantum computation. *Physical Review A*, 69(6):062320, 2004.

[9] Vincent Bouchiat, D Vion, Ph Joyez, D Esteve, and MH Devoret. Quantum coherence with a single cooper pair. *Physica Scripta*, 1998(T76):165, 1998.

[10] Katherine L Brown, William J Munro, and Vivien M Kendon. Using quantum computers for quantum simulation. *Entropy*, 12(11):2268–2307, 2010.

[11] M Büttiker. Zero-current persistent potential drop across small-capacitance josephson junctions. *Physical Review B*, 36(7):3548, 1987.

[12] Chris Cesare. Online security braces for quantum revolution. *Nature*, 525(7568):167–168, 2015.

[13] Livio Ciorciaro. Automatic single qubit routines, 2015.

[14] John Clarke and Frank K Wilhelm. Superconducting quantum bits. *Nature*, 453(7198):1031–1042, 2008.

[15] Altera Corporation. FPGA photo. https://upload.wikimedia.org/wikipedia/commons/f/fa/Altera_StratixIVGX_FPGA.jpg.

[16] Louis De Broglie. Waves and quanta. *Nature*, 112:540, 1923.

[17] Michel H Devoret, Andreas Wallraff, and John M Martinis. Superconducting qubits: A short review. *arXiv preprint cond-mat/0411174*, 2004.

[18] David P DiVincenzo et al. The physical implementation of quantum computation. *arXiv preprint quant-ph/0002077*, 2000.

[19] Albert Einstein. Über einen die erzeugung und verwandlung des lichtes betreffenden heuristischen gesichtspunkt. *Annalen der physik*, 322(6):132–148, 1905.

[20] Artur K Ekert. Quantum cryptography based on bell's theorem. *Physical review letters*, 67(6):661, 1991.

[21] Andrew Fursman. What can quantum computing do for us? https://www.weforum.org/agenda/2015/08/qa-what-quantum-computing-can-do-for-us/.

[22] Christopher Gerry and Peter Knight. *Introductory quantum optics*. Cambridge university press, 2005.

[23] Lov K Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 212–219. ACM, 1996.

[24] Samuel Haberthür. Randomized benchmarking of two-qubit gates. Master's thesis, Masther Thesis, Eidgenössische Technische Hochschule ETH Zürich, 2015, 2015.

[25] Erwin L Hahn. Spin echoes. *Physical review*, 80(4):580, 1950.

[26] Ronald Hanson and David D Awschalom. Coherent manipulation of single spins in semiconductors. *Nature*, 453(7198):1043–1049, 2008.

[27] Johannes Heinsoo. Automatic multi-qubit gate calibration, 2013.

[28] Werner Heisenberg. Über den anschaulichen inhalt der quantentheoretischen kinematik und mechanik. *Zeitschrift für Physik*, 43(3-4):172–198, 1927.

[29] Serge Horoche and Daniel Kleppner. Cavity quantum electrodynamics. *Phys. Today*, 42(1):24, 1989.

[30] D-Wave Systems Inc. Applications of a Quantum Computer. http://www.dwavesys.com/quantum-computing/applications.

[31] National Instruments. LabView icon. http://southafrica.ni.com/sites/default/files/labview-logo.jpg.

[32] Jonathan A Jones and Michele Mosca. Implementation of a quantum algorithm on a nuclear magnetic resonance quantum computer. *The Journal of chemical physics*, 109(5):1648–1653, 1998.

[33] Brian David Josephson. Possible new effects in superconductive tunnelling. *Physics letters*, 1(7):251–253, 1962.

[34] David Kielpinski, Chris Monroe, and David J Wineland. Architecture for a large-scale ion-trap quantum computer. *Nature*, 417(6890):709–711, 2002.

[35] Jens Koch, M Yu Terri, Jay Gambetta, Andrew A Houck, DI Schuster, J Majer, Alexandre Blais, Michel H Devoret, Steven M Girvin, and Robert J Schoelkopf. Charge-insensitive qubit design derived from the cooper pair box. *Physical Review A*, 76(4):042319, 2007.

[36] Paul C Lauterbur. Image formation by induced local interactions: examples employing nuclear magnetic resonance. 1973.

[37] Chris Monroe, DM Meekhof, BE King, WM Itano, and DJ Wineland. Demonstration of a fundamental quantum logic gate. *Physical review letters*, 75(25):4714, 1995.

[38] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge university press, 2010.

[39] Wolfgang Pauli. Über den zusammenhang des abschlusses der elektronengruppen im atom mit der komplexstruktur der spektren. *Zeitschrift für Physik A Hadrons and Nuclei*, 31(1):765–783, 1925.

[40] Max Planck. On the law of distribution of energy in the normal spectrum. *Annalen der Physik*, 4(553):1, 1901.

[41] Norman F Ramsey. A molecular beam resonance method with separated oscillating fields. *Physical Review*, 78(6):695, 1950.

[42] Rohde and Schwarz. MWG photo. http://mwrf.com/site-files/mwrf.com/files/gallery_images/SGU_46945_538ef94600c65.jpg?1447168029.

[43] Y. Salathé, M. Mondal, M. Oppliger, J. Heinsoo, P. Kurpiers, A. Potočnik, A. Mezzacapo, U. Las Heras, L. Lamata, E. Solano, S. Filipp, and A. Wallraff. Digital quantum simulation of spin models with circuit quantum electrodynamics. *Phys. Rev. X*, 5:021027, Jun 2015.

[44] Erwin Schrödinger. Quantisierung als eigenwertproblem. *Annalen der physik*, 385(13):437–490, 1926.

[45] David Isaac Schuster. *Circuit quantum electrodynamics*. 2007.

[46] Eyob A Sete, John M Martinis, and Alexander N Korotkov. Quantum theory of a bandpass purcell filter for qubit readout. *Physical Review A*, 92(1):012325, 2015.

[47] Peter W Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM review*, 41(2):303–332, 1999.

[48] Michael Tinkham. *Introduction to superconductivity*. Courier Corporation, 1996.

[49] Andreas Wallraff, David I Schuster, Alexandre Blais, L Frunzio, R-S Huang, J Majer, S Kumar, Steven M Girvin, and Robert J Schoelkopf. Strong coupling of a single photon to a superconducting qubit using circuit quantum electrodynamics. *Nature*, 431(7005):162–167, 2004.

[50] Inc. Wolfram Research. Number Field Sieve. http://mathworld.wolfram.com/NumberFieldSieve.html.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Spectroscopy Automation and Sample Characterization of Superconducting Qubits

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Storz	Simon

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 19. Dezember 2016	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.