Frequency synthesis and pulse shaping for quantum information processing with trapped ions

diploma thesis in physics

by

Philipp Schindler

submitted to the Faculty of Mathematics, Computer Science and Physics of the University of Innsbruck

in partial fulfillment of the requirements for the degree of Magister der Naturwissenschaften

supervisor: Prof. Rainer Blatt
Institut für Experimentalphysik

September 22, 2008

Contents

1 Introduction **7**
1.1 Quantum information processing . . . 7
1.2 Quantum information processing with trapped ions . . . 9

2 Laser Ion interaction **15**
2.1 The optical Bloch equations . . . 15
2.1.1 The interaction Hamiltonian: . . . 15
2.1.2 Solution without interaction: . . . 16
2.1.3 Solution with constant interaction strength: . . . 16
2.1.4 Off-resonant excitations . . . 18
2.1.5 Solutions for amplitude-shaped laser pulses . . . 19
2.1.6 The influence of the initial state . . . 21
2.1.7 Qubit errors . . . 22
2.2 The influence of pulse shapes . . . 23
2.3 Numerical calculations . . . 27
2.4 The AC Stark effect . . . 34

3 Experimental setup **37**
3.1 The ion trap . . . 37
3.2 Optical setup . . . 37
3.2.1 Lasers . . . 37
3.2.2 The detection system . . . 39
3.3 Experiment control . . . 39
3.4 The programmable pulse generator . . . 42
3.4.1 Overview . . . 42
3.4.2 Frequency Generation . . . 44
3.4.3 The FPGA Core . . . 48
3.5 The Software . . . 51
3.5.1 Overview . . . 51
3.5.2 The Python server . . . 51
3.5.3 Limitations . . . 52
3.6 Future development of the programmable pulse generator . . . 53

4 Experimental results **55**
4.1 Functionality tests of the programmable pulse generator . . . 55
4.1.1 Spectrum of the direct digital synthesizer output . . . 56
4.1.2 Characterization of the phase coherent switching . . . 57
4.1.3 Calibration of the radio frequency power . . . 59

4.2 Starting up and calibrating the experiment 61
4.3 Characterizing off-resonant excitations 63
4.3.1 The effect of pulse shaping 63
4.3.2 The influence of the initial state 67
4.4 CNOT process tomography 68

5 Conclusions and Outlook **71**

Appendix **73**

A Abbreviations **73**

B Programmable pulse generator manual **75**
B.1 The python server 75
B.1.1 Installing the python server 75
B.1.2 starting up the python server 75
B.1.3 Troubleshooting 76
B.1.4 Pulse programming reference 77
B.1.5 Sequential programming 77
B.1.6 Parallel environment 81
B.2 Configuring the software 82
B.2.1 Basic configuration 82
B.2.2 In depth configuration 82
B.2.3 Configuring the devices 84
B.3 Configuring the Hardware 84
B.4 The LabView interface 85
B.4.1 Parallel environment 85
B.4.2 Sequential environment 85
B.4.3 Creating pulse commands 86
B.4.4 Returning values to LabView and using global variables 88
B.4.5 User function framework 88
B.5 Internals of the Software 89

C Internals of the Programmable pulse generator **90**
C.1 The firmware 90
C.2 Pin configuration of the LVDS Bus 97

D Matlab source code **98**
D.1 Simulation of a 2 level system 98
D.2 Calculation of the adiabatic factor 100

E Bibliography **101**

Abstract

This thesis reports on the setup of a new experimental control system for quantum information experiments with trapped ions. This system is used for coherent manipulation of an optical qubit in $^{40}Ca^{+}$ ions. It is capable to generate phase coherent radio frequency pulses with arbitrary amplitude envelopes. The principles of the frequency synthese technique as well as the principle of operation of the entire system are explained.

The effect of amplitude shaped laser pulses on single qubit quantum operations is discussed theoretically and experimentally. Finally an improvement of the fidelity of a controlled not quantum gate due to the use of amplitude shaped laser pulses is shown.

1 Introduction

1.1 Quantum information processing

Quantum information processing has been a major field of investigation in the last 10 years. Whilst some initial proof of concept experiments have been carried out, the realization of a quantum computer with computation power comparable to present computers is in the far future [1]. A classical computer obeys the well understood laws of classical mechanics whereas a quantum computer utilizes physical processes unique to quantum mechanics. In quantum mechanics the classical unit of information, the bit, is replaced by the quantum mechanical analogue called "qubit". A qubit could, in principle, be any quantum mechanical two level system. One of the advantages of quantum computation over classical computation is that for some problems algorithms were found which scale more favorable with growing size of the problem than known classical algorithms. The most famous quantum algorithm is Shor's factorizing algorithm, which is able to factorize efficiently an integer into prime numbers. This is important because this could be used for breaking the security of RSA key encryption which is widely used [2]. Another notable application for a future quantum computer would be simulation of quantum systems, such as the appearance of phase transitions in Ising spin chains. The time required for simulating such quantum systems increases exponentially with the number of qubits on classical computers, whereas in 1982 Richard Feynman showed that it is possible to implement such simulations more efficiently with a quantum computer [3]. The number of qubits required to achieve the same computation power as todays classical computers with Shor's factorizing algorithm lies in the range of 10^5, whereas only around 30 qubits are required for simulating certain many body quantum systems [4, 5].

The requirements for a physical implementation for quantum information processing were given by by DiVincenzo [6]. A system suited for quantum information processing should fulfil the following requirements:

1. The system should be scalable with well characterized qubits.
2. It should be possible to initialize the qubits in a well defined state.
3. The qubit coherence times should be much longer than the gate time.
4. An universal set of quantum gates should exist.
5. A possibility to measure the state of the qubit should exist.
6. A possibility to inter-convert flying and stationary qubits should exist.
7. A possibility to transmit flying qubits between specified locations should exist.

Most of these criteria have been met in proof of concept experiments but scaling up this system to many qubits is a difficult and active research area [1].

There are numerous physical systems proposed as candidates for quantum information processing, these include: trapped ions, neutral atoms in optical traps, nuclear magnetic

resonance (NMR) with liquids and various semiconductor approaches [7]. The most complex experiments achieved so far in quantum information have been realized with an NMR approach, however with this approach there are no general accepted proposals for scaling the concept to many qubits.

Depending on the physical implementations, different approaches for describing a quantum algorithm are known. In this thesis only the "circuit model" similar to the modeling of classical algorithms with logical gates is discussed. Such a quantum algorithm consists of a sequence of multiple qubit operations, but it has been shown that it is possible to implement any quantum algorithm only with single- and two-qubit operations [8]. Therefore a quantum computer is universal if it is able to implement arbitrary one- and two-qubit operations.

The qubit consisting of the quantum states $|0\rangle$ and $|1\rangle$ may be in the general state $\Psi = \alpha\,|0\rangle + \beta\,|1\rangle$, where α and β are complex numbers and satisfy $|\alpha|^2 + |\beta|^2 = 1$. Any single qubit operation $\hat{U}$ mapping a state Ψ to another state $\Psi' = \hat{U}\Psi$ may be written as a unitary 2x2 matrix U. Analogously a two qubit state may be expressed as $\Psi = \alpha_1\,|00\rangle + \alpha_2\,|01\rangle + \alpha_3\,|10\rangle + \alpha_4\,|11\rangle$, therefore any two-qubit operation may be written as a unitary 4x4 matrix.

It was shown that one universal set of gates is the controlled not gate (CNOT) combined with arbitrary single qubit rotations [8]. The CNOT is a two qubit gate which inverts the value of the target qubit depending on the state of the control qubit. The classical analogue to the CNOT would be the XOR gate. The CNOT gate is also an entangling gate, which means it is possible to generate an entangled output state from a product input state :

$$\frac{1}{\sqrt{2}}(|10\rangle + |00\rangle) \xrightarrow{CNOT} \frac{1}{\sqrt{2}}(|11\rangle + |00\rangle)$$

The CNOT process matrix written in the basis states $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$ is :

$$U = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

In an experimental implementation imperfections may be quantified by obtaining the experimental process matrix and comparing it to the ideal process matrix. With quantum process tomography full information about a quantum process may be retrieved [9]. When the process matrix is known, benchmark values for quantum gates, as the fidelity, may be calculated easily.

With the concept of quantum error correction it is possible to correct for errors induced in the implementation of the quantum algorithms. Similar of classical error correction, the error is detected with the help of auxiliary qubits. For applying quantum error correction

efficiently the error rate of the quantum gates has to be below 10^{-4} [10, 11].

1.2 Quantum information processing with trapped ions

Quantum information processors based on trapped ions are one of the most promising candidates for a future quantum computer. CNOT gates with a fidelity of up to 97% have already been achieved and first experiments with large scale micro-fabricated ion traps have been carried out [5, 12].

In an ion trap quantum information processor, the quantum mechanical system used for storage and manipulation of the quantum information is a set of stable electronic states of the ion. These states may be either a ground state and some metastable excited state, or two ground states which are energetically separated, e.g. by hyperfine splitting. Single-qubit operations are realized with either timed laser or radio frequency pulses depending on the type of the qubit and the level structure of the ion species used. An additional demand is the need for addressing the ions individually. In our setup these operations are realized with a laser which is focused to a spot size smaller than the inter ion distance. Switching the laser between different ions is realized by deflecting the laser beam along the axis of the ion crystal.

The ions are confined in a linear Paul trap which consists of four rods, which are connected either to a radio frequency or a DC voltage, and two end-caps with a DC voltage applied. To achieve well defined ion positions, the ions are cooled with the help of Doppler cooling until a linear ion crystal is formed [13].

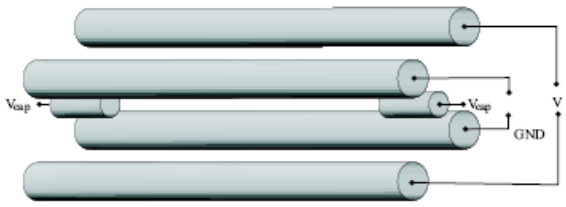

Figure 1: Schematic drawing of a Paul trap.

The trap potential may be approximated by a three dimensional quantum mechanical harmonic oscillator which leads to quantized motion of the ions. The motional quanta of this oscillator are common to all ions and are used to realize multiple qubit interactions [14]. In our setup only the axial motional modes are used. The spectrum of the qubit transition for three ions is shown in Fig. 2 where the center peak corresponds to driving the qubit transition without changing the phonon number. This transition is denoted as the "carrier" transition. The other transitions are denoted as "sideband" transitions, where the phonon count in the common mode is changed while driving one of these transitions. The transition which increases the phonon count is denoted the "blue sideband" transition, whereas the "red sideband" transition decreases the phonon number. For two and more

ions, analogous to a classical oscillator,several motional modes may be excited, however describing this behaviour is beyond the scope of this thesis. A full description of ion crystals is given in the PhD thesis of Hanns Christoph Nägerl [13].

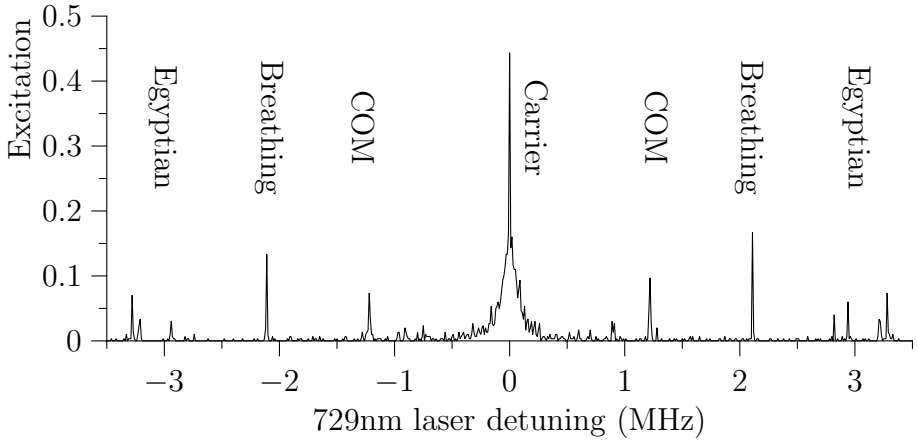

Figure 2: Excitation spectrum of the axial sidebands on the qubit transition for 3 $^{40}Ca^{+}$ ions. The motional modes with positive detuning are blue sideband transitions.

In our research group $^{40}Ca^{+}$ and $^{43}Ca^{+}$ ions are used for quantum information experiments, where the $^{40}Ca^{+}$ qubit is realized with a metastable excited state and the $^{43}Ca^{+}$ qubit is encoded in the hyperfine structure of the ground state. This thesis will concentrate on the $^{40}Ca^{+}$ experiment, but the results presented in this thesis are also applicable to other qubit implementations.

A simplified level scheme of $^{40}Ca^{+}$ containing only the transitions used in our experiment is shown in Fig. 3. The qubit transition is the metastable $S_{1/2} \rightarrow D_{5/2}$ quadrupole transition, which has a lifetime of $\tau \approx 1s$. As the gate time has to be much shorter than this coherence time this leads to a maximum gate time of $\tau_{gate} \leq 10\,ms$. The 866nm , 397nm and 854nm dipole transition are required for initializing and measuring the qubit. For state detection, the electron shelving technique on the $S_{1/2} \rightarrow P_{1/2}$ dipole transition is used [13]. The 866nm laser empties the $D_{3/2}$ state during readout whereas the 854nm laser is used to prepare the ion in the ground state at the beginning of an experiment and for emptying the $D_{5/2}$ state during sideband cooling.

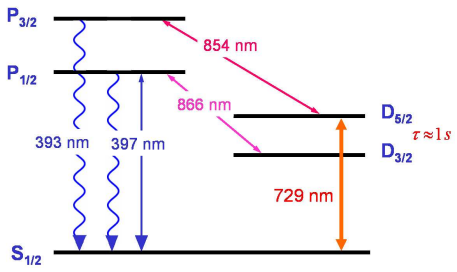

Figure 3: Simplified level scheme of the $^{40}Ca^{+}$ ion. The qubit transition is the 729nm $S_{1/2} \rightarrow D_{5/2}$ transition. The 397nm $S_{1/2} \rightarrow P_{1/2}$ transition is used for Doppler colling, state preparation and detection whereas the 866nm $D_{3/2} \rightarrow P_{1/2}$ and the 854nm $D_{5/2} \rightarrow P_{3/2}$ transition are used for repumping.

The experimental sequence

The qubit is manipulated with timed laser pulses, where the main components are qubit initialization, coherent manipulation and state detection. In order to measure the probability P_D of finding an ion in the $D_{5/2}$ state each sequence is repeated 100-200 times for a single data point. In order to perform qubit operations the ion has to be prepared in a well defined initial state, so the first part of every experimental sequence is cooling the ion via Doppler cooling and preparing them in a well defined Zeeman sub-level with optical pumping. Doppler cooling is achieved with a red detuned 397nm beam. The average phonon number after Doppler cooling is about ten. During Doppler cooling the 866nm laser is shone on the ions to empty the $D_{3/2}$ level. To prepare the system in the motional ground state, sideband cooling is applied, for which the 729nm light is detuned by the ion oscillation frequency to the red, which means that a π pulse on the red sideband decreases the phonon numberby one. An intuitive picture of this process is shown in Fig. 4.

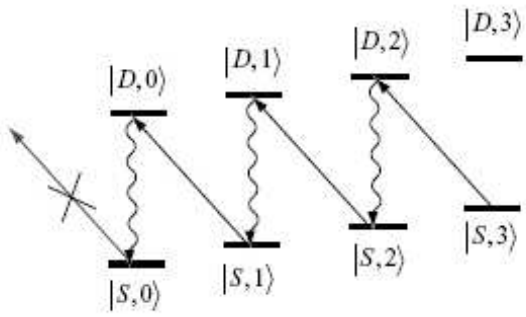

Figure 4: Simplified scheme of sideband cooling. The red sideband $S \rightarrow D$ transition of the ion is driven to decrease the phonon number in the harmonic oscillator. The D state decays then back into the S state with a phonon count decreased by one.

This cooling scheme relies on emission from the $D_{5/2}$ to the $S_{1/2}$ level, but since this transition is metastable the cooling rate would be about 1 phonon per second, which is too slow. Therefore the $D_{5/2}$ level is pumped to the short-lived $P_{5/2}$ level with a 854nm laser. The probability of finding the ion in the motional ground state after sideband cooling is 98% or higher, depending on the number of ions in the crystal.

After cooling, the qubit is coherently manipulated, where precisely timed laser pulses are applied on the qubit transition and its sidebands. By applying a resonant pulse on the carrier transition and varying the pulse length, "Rabi flops", an oscillation between the ground and the excited state, are observed. The length of a pulse may be described as a Rabi phase where a pulse with length 2π corresponds to a full oscillation and leaves the population unchanged.

For state dependent detection, the electron shelving technique is used, which has an efficiency of over 99% [13]. The detection laser at 397nm couples only to the $S_{1/2}$ level, which means that there is no fluorescence light when the ion is in the $D_{5/2}$ state. The state where fluorescent light is observed is denoted as logical one ($|1\rangle$). The number of repetitions

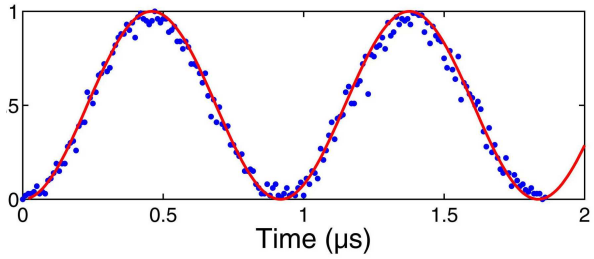

Figure 5: Population oscillations on the $S_{1/2} \rightarrow D_{5/2}$ carrier transition for varying pulse lengths. These population oscillations are called Rabi flops.

for the same sequence determines the projection noise of the detection scheme [15].

Off-resonant excitations

The goal of this thesis is to quantify and minimize the influence of off-resonant excitations to the gate fidelity. The main error sources for a CNOT gate in our experiment, as discussed in the PhD thesis of Mark Riebe, are shown in Tab. 1 [16].

error source	Contribution
laser frequency noise + magnetic field noise	11%
Residual thermic excitation	2%
addressing error	2%
off-resonant excitations	4%

Table 1: Contributions of different experimental imperfections on the CNOT gate error.

Laser frequency noise is reduced by decreasing the 729nm linewidth, which was approximately 1kHz. After improving the locking scheme of the laser, a linewidth below 10Hz in 1s was measured. The magnetic field noise is minimized by constructing a magnetic field shielding box around the vacuum vessel, suppressing magnetic fields by at least a factor of ten. Addressing error may be compensated by applying special correction pulses.

The largest remaining error source is then given by off-resonant excitations. Off-resonant excitations of the carrier transition play a role when a sideband transition is driven, because the coupling strength of the sideband is much weaker than that of the carrier transition. Therefore the optical power has to be increased to achieve acceptable Rabi times on the blue sideband, which on the other hand increases the effect of off-resonant excitations on the carrier transition. The easiest way to decrease the off-resonant excitations is to increase the difference between the carrier and the sideband frequency. However by doing so the inter-ion distance is decreased and therefore the addressing error is increased and sideband cooling becomes less efficient. Another way of minimizing off-resonant excitations is to switch the laser pulse on and off adiabatically, which technically means using amplitude modulated laser pulses. The influence of the pulse form on off-resonant excitations is discussed in this thesis.

In this thesis a basic theory of ion laser interaction is derived. From this theory the influence of amplitude shaped pulses on the behaviour of a single ion is analyzed in section(2.1). A simple method for testing whether a particular pulse suppresses off-resonant excitations is introduced in section(2.2). Numerical simulations for various pulse shapes are presented in section(2.3). A new experiment control setup and a new source for RF pulses are introduced in section(3). The results of the theory are compared with measured data in section(4). The effects of pulse shaping on the fidelity of a quantum gate are demonstrated in section(4.4). The appendix of this thesis consists of technical documentation for the experiment control system.

2 Laser Ion interaction

In the following chapter the interaction of the ion with a laser beam will be examined theoretically. The main emphasize is on quantifying off-resonant excitations and the influence of amplitude-shaped laser pulses. For this purpose a two-level system is investigated. The influence of other levels, the quantized motion, and the spatial profile of the laser beam are neglected. The system is investigated analytically for time-independent interaction strength. For time dependent interaction strength numerical calculations are performed as there exists no general analytical solution. The description shown below follows chapter 3 of reference [17].

2.1 The optical Bloch equations

To characterize the dynamics of the system, the Schrödinger equation has to be solved. The Hamiltonian of the system may be decomposed into a time-independent component H_o and a time-dependent interaction component H_I. The Hamiltonians for a two-level system may generally be written as Hermitian 2x2 matrices, and the state of the system may be expressed as a complex valued two-dimensional vector. When choosing the basis states as eigenstates of H_0, the matrix H_0 will contain no off-diagonal elements.

$$\hat{H} = \hat{H}_0 + \hat{H}_I(t)$$

2.1.1 The interaction Hamiltonian:

The interaction between the ion and the applied electromagnetic field is described by coupling the multipole moments of the ion to the external field [17]. The most dominant interactions are the electric dipole interaction, the magnetic dipole interaction and the electric quadrupole interaction. Independent of the actual interaction type, the interaction strength is expressed as the complex valued Rabi frequency Ω. The dependence of Ω on the amplitude of the driving field E is shown in Tab. 2, where μ is the electric dipole moment, q the magnetic quadrupole moment and μ_m, the magnetic dipole moment.

Interaction type	Rabi frequency
electric dipole	$\Omega_d = \frac{\mu E}{\hbar}$
electric quadrupole	$\Omega_q = \frac{q \bigtriangledown E}{\hbar}$
magnetic dipole	$\Omega_m = \frac{\mu_m B}{\hbar}$

Table 2: Definition of the Rabi frequency for different interaction types.

When the motion of the ion is neglected this leads to the following interaction Hamiltonian:

$$\hat{H}_I(t) = \begin{pmatrix} 0 & \hbar\Omega(t)\ \sin(\omega t + \phi) \\ \hbar\Omega^*(t)\ \sin(\omega t + \phi) & 0 \end{pmatrix} \tag{1}$$

For a general amplitude-shaped laser pulse the Rabi frequency $\Omega(t)$ is time-dependent and complex valued. Depending on the actual type of interaction, the polarization of the

laser light and the choice of the basis the Rabi frequency and the oscillatory term may be complex or real valued. In this thesis the laser frequency is constant during a laser pulse and therefore it is possible to choose a basis such that the Rabi frequency and the oscillatory term are real valued.

2.1.2 Solution without interaction:

It is assumed that the wave function Ψ fulfils the Schrödinger equation for the time-independent Hamiltonian $\hat{H}_0$. This means the equation $\hat{H}_0\Psi = i\hbar\delta_t\Psi$ is fulfilled. Any quantum mechanical two-level system may be fully described by a basis of two eigenstates $\Psi_{1,2}$ of the Hamiltonian with the respective energy eigenvalues $E_{1,2}$. The transition frequency is then defined as $\omega_0 = (E_2 - E_1)/\hbar$. Every possible state Ψ may then be written as

$$\Psi(\mathbf{r},t) = \tilde{C}_1\Psi_1(\mathbf{r},t) + \tilde{C}_2\Psi_2(\mathbf{r},t)\ . \tag{2}$$

Since the Hamiltonian is time independent, $\Psi(\mathbf{r},t)$ may then be factorized into a product of a time dependent and a time independent function.

$$\begin{aligned} \Psi_1(\mathbf{r},t) &= \exp(-iE_1t/\hbar)\,\psi_1(\mathbf{r}) \\ \Psi_2(\mathbf{r},t) &= \exp(-iE_2t/\hbar)\,\psi_2(\mathbf{r}) \end{aligned} \tag{3}$$

In the following derivation the space dependence of Ψ will be neglected as the electric field amplitude is assumed to be constant in space.

2.1.3 Solution with constant interaction strength:

As described above $\Psi_{1,2}$ are a set of orthogonal basis vectors and therefore the solution for any Hamiltonian may be written as $\Psi = \tilde{C}_1(t)\Psi_1(t) + \tilde{C}_2(t)\Psi_2(t)$. For describing the interaction between the laser light and the ion, the Hamiltonian which is defined in Eq. 1 is used. As the frequency of driving laser field is ω, a detuning between the laser frequency and the transition frequency may be defined as $\delta = \omega - \omega_0$. A schematic view of the involved levels and frequencies is given in Fig. 6. When inserting this ansatz into the Schrödinger equation this leads to a set of coupled differential equations for $C_{1,2}$.

$$i\hbar\frac{d}{dt}\begin{pmatrix}\tilde{C}_1(t)\\ \tilde{C}_2(t)\end{pmatrix} = \begin{pmatrix} E_1 & \hbar\Omega(t)\,\sin(\omega t+\phi) \\ \hbar\Omega(t)\,\sin(\omega t+\phi) & E_1+\hbar\omega_0\end{pmatrix}\begin{pmatrix}\tilde{C}_1(t)\\ \tilde{C}_2(t)\end{pmatrix}$$

The solution is simplified if the equations are expressed in a time-dependent basis with the same phase evolution as the laser field. This is realized by introducing new coefficients $C_{1,2}(t)$ which differ from $\tilde{C}_{1,2}$ by a continuous evolving phase:

$$C_{1,2}(t) = \tilde{C}_{1,2}(t)\,e^{-i\eta_{1,2}(t)}$$

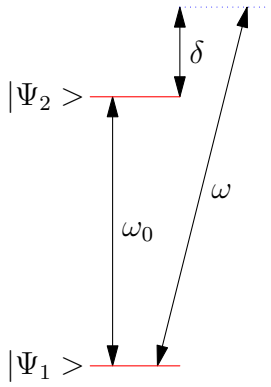

Figure 6: Level scheme including a detuned driving laser field.

The state vectors in the rotating wave picture are then written as

$$\Psi = C_1(t)e^{-i\eta_1(t)}\Psi_1(t) + C_2(t)e^{-i\eta_2(t)}\Psi_2(t) \; .$$

The choice of $\eta_1(t) = (E_1/\hbar - \delta/2)t$ and $\eta_2(t) = \eta_1(t) + \omega t + \phi$ leads to the following equations:

$$i\,\frac{d}{dt}\begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} = \frac{1}{2}\begin{pmatrix} \delta & \Omega(1+e^{-2i(\omega t+\phi)}) \\ \Omega(1+e^{2i(\omega t+\phi)}) & -\delta \end{pmatrix}\begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix}$$

The particular choice of $\eta_{1,2}$ resembles a frame of reference which rotates with the laser frequency ω. The resulting differential equations now shows diagonal elements with the small quantity $\pm\delta/2$. The terms oscillating with the laser frequency ω may be approximated by their cycle average if $\delta \ll 2\omega$ and $\Omega \ll \omega_0$. These conditions are generally well satisfied for optical transitions. As the cycle average of $\exp(-2i\big(\omega t + \phi\big))$ is zero this leads to the following simplified differential equation:

$$i\,\frac{d}{dt}\begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} \approx \frac{1}{2}\begin{pmatrix} \delta & \Omega \\ \Omega & -\delta \end{pmatrix}\begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix} \tag{4}$$

As a first step, the Rabi frequency will be assumed to be constant in time for the following analysis. In section(2.3) numerical simulations with time varying $\Omega(t)$ are presented. Eq.4 may be solved analytically by replacing the first order differential equation with the two variables $C_{1,2}(t)$ with a second order differential equation with only one variable. The results are combinations of sines and cosines with the oscillation frequency $\tilde{\Omega}/2$ where $\tilde{\Omega} = \sqrt{\Omega^2 + \delta^2}$. The amplitudes of the sines and cosines are determined by the initial conditions

$$C_{1,2}(t) = A_{1,2}\cos(\frac{\tilde{\Omega}t}{2}) + B_{1,2}\sin(\frac{\tilde{\Omega}t}{2}) \; .$$

If the system is initially in the ground state, the populations $P_{1,2} = |C_{1,2}|^2$ are:

$$\begin{aligned} P_1(t) &= |\frac{\Omega}{\tilde{\Omega}}|^2 \sin^2(\tilde{\Omega}t) \\ P_2(t) &= 1 - P_1(t) \quad . \end{aligned} \tag{5}$$

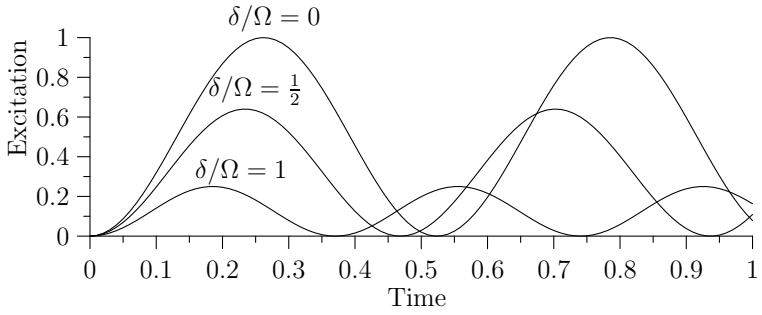

Figure 7: Rabi Oscillations for a rectangular laser pulse at detuning δ/Ω.

The results obtained in Eq.5 shows oscillations in the population of the two levels, which are called Rabi oscillations. In Fig. 7 these oscillation are shown for different parameters.

2.1.4 Off-resonant excitations

As described in section(1.2) it is necessary to excite the motional sideband in order to perform multiple-qubit operations in an ion trap quantum information processor. However, the coupling strength on the sideband transition is smaller than that on the carrier transition by the Lamb-Dicke factor which is in our setup about 3% [16]. Therefore the sideband transition has to be excited with higher laser power to achieve an acceptable Rabi frequency. This leads to unwanted changes in the state vector of the carrier transition during a laser pulse on the sideband transition. These changes vary with the duration of the interaction and may be divided into phase and population deviations. The changes in the qubit population are known as off-resonant excitations whereas the phase variations are described by the AC Stark effect. The AC Stark effect describes a frequency shift of the carrier transition, which gives rise to a dephasing of the qubit vector. This effect is discussed separately in section(2.4) whereas the following paragraphs will concentrate on the population changes.

From equation (5) it can be seen that for a rectangular pulse the magnitude of these population oscillations is $\Omega^2/(\Omega^2 + \delta^2)$ when the system was initially in the ground state. It can also be seen that for a constant Rabi frequency the off-resonant excitations vanish with large laser detuning. Fig. 8 shows populations oscillations that occur on the carrier when driving a sideband pulse. Our usual experimental parameters for driving sideband transitions are $\Omega = 2\pi\,200\,\mathrm{kHz}$ and $\delta = 2\pi\,1\,\mathrm{MHz}$. This leads to population oscillations with an amplitude of 3.8%. As our CNOT gate fidelity reaches more than 90% these oscillations have to be minimized or compensated.

A possible strategy for compensating off-resonant excitations would be to switch off the laser when the induced population oscillation returns to zero. However, in our experimental implementation this is not practical as the oscillation frequency increases with the detuning and therefore the calculation and generation of the pulses becomes impractical. The detuning from the carrier transition is $\delta \approx 2\pi 1\,\mathrm{MHz}$ when driving the blue sideband transition. As the Rabi frequency is only $\Omega = 2\pi 200\,\mathrm{kHz}$ the population oscillates with the effective Rabi frequency $\tilde{\Omega} \approx \delta = 2\pi 1\,\mathrm{MHz}$. The length of a typical sideband pulse is around $100\,\mu\mathrm{s}$ which means that the desired minimum is after 100 oscillation periods. Therefore using this technique would require a very precise knowledge of the Rabi frequency as even a 1% error would correspond to a whole oscillation period of these oscillation and would therefore make the compensation unusable. Furthermore the experimental parameters would need to be stable over a long timescale to ensure proper compensation of these off-resonant excitations without frequent recalibration. Our approach instead is to switch on the interaction adiabatically which eliminates off-resonant excitations. This technique is discussed in section(2.2)

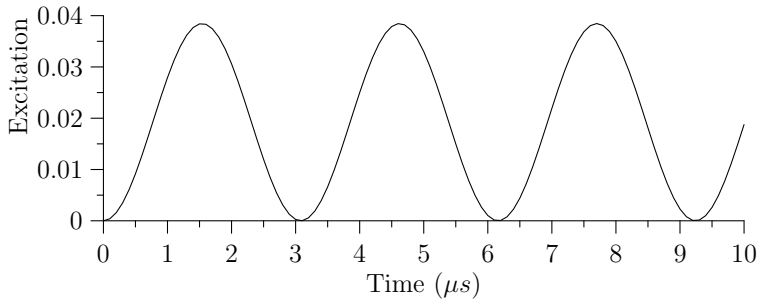

Figure 8: Calculated off-resonant excitations on the carrier while driving a sideband pulse. The parameters are $\Omega = 2\pi\,200\,\mathrm{kHz}$ and $\delta = 2\pi\,1\,\mathrm{MHz}$.

2.1.5 Solutions for amplitude-shaped laser pulses

For the results presented above the system was considered to be initially in the ground state $\Psi(0) = \Psi_1$ and the Rabi frequency was constant over time. In this section the results are generalized to arbitrary input states and time dependent Rabi frequencies. To calculate the time evolution for an arbitrary initial state $\Psi(0) = \gamma_1\Psi_1 + \gamma_2\Psi_2$, the system is rotated to basis states of the particular Hamiltonian. These states are known as dressed states and the corresponding basis is denoted as the dressed basis. The new basis states $\Phi_\pm$ are defined as

$$\begin{pmatrix} \Phi_+ \\ \Phi_- \end{pmatrix} = \begin{pmatrix} \cos\beta & -\sin\beta \\ \sin\beta & \cos\beta \end{pmatrix} \begin{pmatrix} \Psi_1 \\ \Psi_2 \end{pmatrix} \tag{6}$$

The states $\Phi_\pm$ are orthogonal and may be chosen as a time dependent basis. In this basis the state vector may be rewritten as

$$\Psi(t) = A_+(t)\,\Phi_+(t) + A_-(t)\,\Phi_-(t) \tag{7}$$

To facilitate the task of constructing eigenvectors of the Schrödinger equation the mixing angle β is introduced:

$$\cos(2\beta) = \frac{\delta}{\tilde{\Omega}} = \frac{\delta}{\sqrt{\Omega^2 + \delta^2}} \tag{8}$$

$$\sin(2\beta) = \frac{|\Omega|}{\tilde{\Omega}} = \frac{|\Omega|}{\sqrt{\Omega^2 + \delta^2}} \tag{9}$$

For a time dependent Rabi frequency $\Omega(t)$ this transformation is made at every time t. This leads to a time dependent mixing angle $\beta(t)$ and in consequence the Schrödinger equation then reads [17]

$$\frac{d}{dt}\begin{pmatrix} A_+(t) \\ A_-(t) \end{pmatrix} = -\frac{i}{2}\begin{pmatrix} \tilde{\Omega}(t) & 2i\dot{\beta}(t) \\ -2i\dot{\beta}(t) & -\tilde{\Omega}(t) \end{pmatrix}\begin{pmatrix} A_+(t) \\ A_-(t) \end{pmatrix}.$$

The quantities which determine the time evolution of the system are $\tilde{\Omega}$ and $\dot{\beta}$. The contributions of these may be interpreted as follows:

- $\tilde{\Omega}$ shifts the energy levels which leads to a continuous phase evolution which is described by the AC-Stark effect.

- $\dot{\beta}$ gives rise to population changes between $A_\pm$which are usually referred to as to off-resonant excitations.

If $\dot{\beta}$ is small compared to $\tilde{\Omega}$ the off diagonal terms in the equation above can be neglected. This approximation is known as the adiabatic approximation. In the experiment this is realized by switching the interaction on slowly. The condition for this is $2i\dot{\beta}(t) \ll \tilde{\Omega}(t)$ which may be rewritten as [17]:

$$|\dot{\Omega}(t)\,\delta| \ll |\tilde{\Omega}(t)|^3 \tag{10}$$

As the transformed Schrödinger equation now contains only diagonal elements it is possible to simply integrate the single components. The solutions read then

$$A_\pm(t) \approx \exp\left(\mp\frac{i}{2}\int_0^t dt'\,\tilde{\Omega}(t')\right) A_\pm(0)\,. \tag{11}$$

From this result it is clear that $|A_\pm(t)|^2 = |A_\pm(0)|^2$. A laser pulse shape which is typically used in our experiment is shown in Fig.(9). For these shapes, the Rabi frequency vanishes at the beginning and the end of the pulse. This leads to $\beta = 0$ and therefore the matrix in Eq. 6 is the unity matrix. This means that the dressed basis is equivalent to the the original basis if the interaction vanishes and therefore $A_\pm(0) = C_{1,2}(0)$ and $A_\pm(t_{end}) = C_{1,2}(t_{end})$. Therefore, the initial and the final states are identical in the original basis state $|C_{1,2}(t_{end})|^2 = |C(0)|^2$ which means that there are no remaining off-resonant excitations.

The phase of the state however evolves with the integral over the effective Rabi frequency. This phase oscillation is caused by the AC-Stark effect which is described in section(2.4).

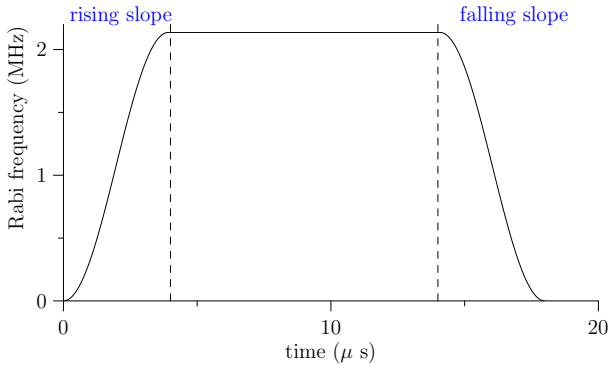

Figure 9: Typical Laser pulse as used in our experiment. The pulse is split into three parts: the rising slope, the constant plateau and the falling slope.

2.1.6 The influence of the initial state

The time evolution of a rectangular pulse for an arbitrary initial state can be calculated analytically. At the leading and the final edge of the pulse the time derivative of the Rabi frequency and therefore also $\dot{\beta}$ is not defined. The time evolution may then be calculated with the following procedure:

1. Transform the initial state at the leading edge to the dressed basis.
2. Calculate the time evolution in the dressed basis.
3. Transform the final state at the final edge back to the original basis.

In contrast to the situation of shaped pulses the Rabi frequency does not vanish at the edges of the pulse. Therefore the dressed state and the original state are not equal. As the Rabi frequency is constant during the pulse duration, the time evolution in the dressed basis consists of a linear phase evolution. Since the dressed basis is not the original basis at the beginning and the end of the pulse this phase evolution causes population oscillations in the original basis. Summarizing these steps leads to the following equation:

$$\begin{aligned}\Psi(t) = &\begin{pmatrix} \cos\beta & \sin\beta \\ \sin\beta & \cos\beta \end{pmatrix} \begin{pmatrix} e^{\frac{i}{2}\tilde{\Omega}t} & 0 \\ 0 & e^{-\frac{i}{2}\tilde{\Omega}t} \end{pmatrix} \cdot \\ &\begin{pmatrix} \cos\beta & \sin\beta \\ \sin\beta & \cos\beta \end{pmatrix} \begin{pmatrix} C_1(0)\,\Psi_1 \\ C_2(0)\,\Psi_2 \end{pmatrix}\end{aligned} \tag{12}$$

With the definition of β as shown in Eq. 8 this equation may be simplified to

$$\Psi(t) = e^{-i\delta/2} \begin{pmatrix} \cos(\frac{\tilde{\Omega}}{2}t) + i\,\frac{\delta}{\tilde{\Omega}}\,\sin(\frac{\tilde{\Omega}}{2}t) & -i\,\frac{|\Omega|}{\tilde{\Omega}}\,\sin(\frac{\tilde{\Omega}}{2}t) \\ -i\,\frac{|\Omega|}{\tilde{\Omega}}\,\sin(\frac{\tilde{\Omega}}{2}t) & \cos(\frac{\tilde{\Omega}}{2}t) - i\,\frac{\delta}{\tilde{\Omega}}\,\sin(\frac{\tilde{\Omega}}{2}t) \end{pmatrix} \begin{pmatrix} C_1(0)\,\Psi_1 \\ C_2(0)\,\Psi_2 \end{pmatrix}. \quad (13)$$

For further analyzing the initial state is assumed to be $\Psi(0) = \Phi_- = \Psi_1 \cos\theta - \Psi_2 \sin\theta$ with the mixing angle θ. The populations $|C_{1,2}(t)|^2$ may then be calculated:

$$|C_{1,2}(t)|^2 = |C_{1,2}(0)|^2 + A_{1,2}\,\sin^2(\frac{\tilde{\Omega}t}{2}) \quad (14)$$

Where the coefficients $A_{1,2}$ correspond to the magnitude of the population oscillations and therefore also describe the magnitude of the off-resonant excitations. For example, the coefficient A_{12} can be simplified to:

$$A_1 = \frac{\frac{\Omega^2}{\delta^2}\cos(2\theta) - \frac{\Omega}{\delta}\sin(2\theta)}{1 + \frac{\Omega^2}{\delta^2}} \quad (15)$$

From this equation it can be seen that the magnitude of the population oscillations is not constant with varying mixing angle θ. The excitations are shown in Fig. 10. It can also be seen that there exist a certain mixing angle where there are no remaining population oscillations. This situation corresponds to the initial state being the dressed state of the system.

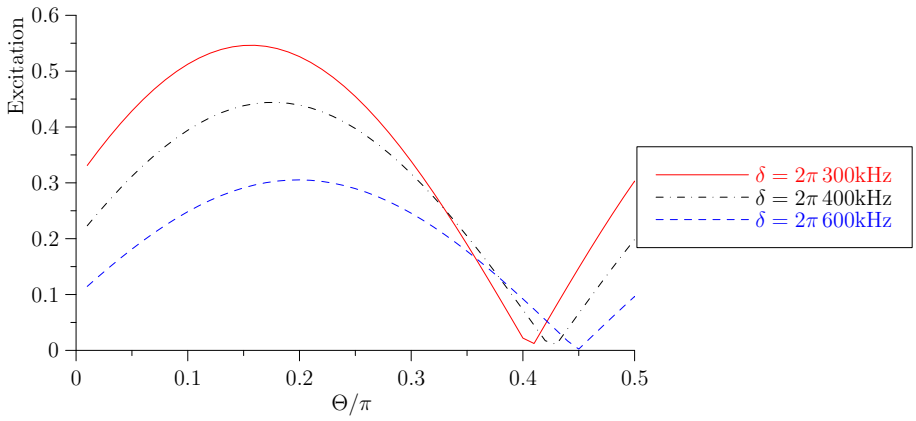

Figure 10: Off-resonant excitation depending on the initial state. The initial state was of the form $\Psi = \cos(\theta)\ |\Psi_1\rangle + \sin(\theta)\,|\Psi_2\rangle$.

2.1.7 Qubit errors

The results presented above show that the magnitude of the population oscillations depends on the initial state of the qubit. Therefore this magnitude is not a good measure for characterizing off-resonant excitations. An error on the qubit may be characterized by the error of the mixing angle. A small error angle ϵ on the qubit vector $|\Psi_{id}\rangle = \cos(\theta)\ |\Psi_1\rangle +$

$\sin(\theta)\ |\Psi_2\rangle$ leads to the vector

$$|\Psi_{err}\rangle = \cos(\theta+\epsilon)\ |\Psi_1\rangle + \sin(\theta+\epsilon)\ |\Psi_2\rangle\ .$$

A second order Taylor expansion leads to the following relation for the population of the ground state:

$$\begin{aligned} p = |\,\langle\Psi_{err}\,|\,\Psi_1\rangle\,|^2 &\approx \left(\cos(\theta) - \sin(\theta)\,\epsilon - \frac{1}{2}\cos(\theta)\,\epsilon^2\right)^2 = \\ &= \cos^2(\theta) - \sin(2\theta)\,\epsilon + \left(1 - 2\,\cos^2(\theta)\right)\epsilon^2 + \mathcal{O}(\epsilon^3)\ . \end{aligned} \tag{16}$$

If an atomic transition is driven off-resonantly, the state vector oscillates with the effective Rabi frequency $\tilde{\Omega}$ which can be seen in Eq. 14. The phase error ϵ also oscillates with the same frequency.

To generalize the characterization of off-resonant excitations for an arbitrary mixing angle Θ the results shown in Eq. 16 are compared to the results derived for a rectangular pulse as shown in Eq. 14. The derivation for a general qubit error has been made under the assumption $\epsilon \ll 1$. In the case of the rectangular pulses this corresponds to a small Rabi frequency with respect to the detuning ($\Omega \ll \delta$). Therefore the result in Eq. 15 may be approximated to

$$A_1 \approx -\frac{\Omega}{\delta}\sin(2\Theta) + \frac{\Omega^2}{\delta^2}\cos(2\Theta) + \mathcal{O}(\frac{\Omega^3}{\delta^3})\ .$$

For a general qubit error this reads

$$p_1 - \cos(\theta) = -\epsilon\,\sin(2\Theta) + \epsilon^2\,\cos(2\Theta)\}\ .$$

Therefore it is valid to identify the error angle ϵ with the ratio between the Rabi frequency and the detuning (Ω/δ). In the literature off-resonant excitations are usually characterized by the amplitude of the population oscillations if the system is initially in the ground state. For a rectangular pulse this amplitude is just Ω^2/δ^2. To be consistent with the usual definition of off-resonant excitations it is desirable to describe the error for an arbitrary mixing angle by ϵ^2 rather by ϵ. If the amplitude of the population oscillations is A_{pop} this leads to following expressions for the amount of the off-resonant excitations:

$$\epsilon^2 = \begin{cases} \frac{A_{pop}^2}{\sin^2(2\Theta)} & \text{if } \sin(2\Theta) \gg \epsilon \\ A_{pop} & \text{if } \Theta = 0 \end{cases} \tag{17}$$

2.2 The influence of pulse shapes

In the following section different laser intensity pulse shapes and their influence on off-resonant excitations are discussed. A suitable shape should minimize off-resonant excitations while keeping the time needed for realizing a certain pulse on the sideband transition

as close as possible to a rectangular pulse with the same peak Rabi frequency.

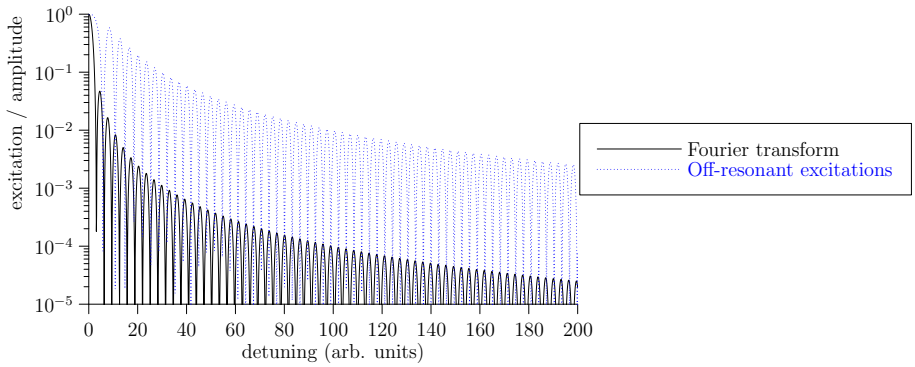

Figure 11: Comparison of the spectra of a discrete Fourier transform of a sine with a rectangular windowing function and off-resonant excitations for a rectangular pulse. The off-resonant excitations are obtained for a laser pulse with constant length and varying detuning.

As a starting point for a pulse shape, windowing functions used in discrete Fourier transformation are used. The spectra of the Fourier transformation of a sine function over a finite time interval and the off-resonant excitation of a rectangular laser pulse with constant length are shown in Fig. 11. The off-resonant excitations for a pulse with unity length as a function of the detuning is $\Omega^2/(\Omega^2+\delta^2)\,\sin^2(\sqrt{\Omega^2+\delta^2})$ whereas for a Fourier transformation with unity length time window this is $\sin^2(\delta)/\delta^2$. For large detunings these two functions behave the same when neglecting a constant factor and therefore the side lobes in the Fourier spectrum are related to off-resonant excitations. Furthermore, a technique to suppress the side lobes of the Fourier transform is therefore also useful for minimizing off-resonant excitations. In digital signal processing so called "windowing functions" are used to suppress these side lobes [18]. Windowing functions add weights to the data points depending on the position in the time window. The side lobes may be interpreted as edge effects caused by the finite length time window. Therefore windowing functions generally vanish smoothly at the beginning and the ending of the time window to minimize these edge effects. Our approach is to use the well known windowing functions for minimizing off-resonant excitations while driving sideband transitions. In digital signal processing the windowing function is chosen to match the expected signal form. A general purpose windowing function is the approximated three-term Blackman[1] function:

$$w(x) = \frac{1}{2}\left(0.84 - \cos(x\pi) + 0.16\,\cos(2x\pi)\right)$$

As mentioned above, a suitable pulse shape should not only be able to suppress off-resonant excitations but also preserve the time needed for a certain pulse on the sideband transition. The peak Rabi frequency of the pulse is limited by thermal effects in the acousto-optical modulator which generates the laser pulses. Therefore, the pulse area of the shaped pulse has to be similar to the pulse area of a rectangular pulse with the

[1] The term "Blackman function" is ambiguously used in the literature. In this work the definitions from reference [18] are used. In the following the approximated three-term Blackman windowing function is always referred to as the Blackman function.

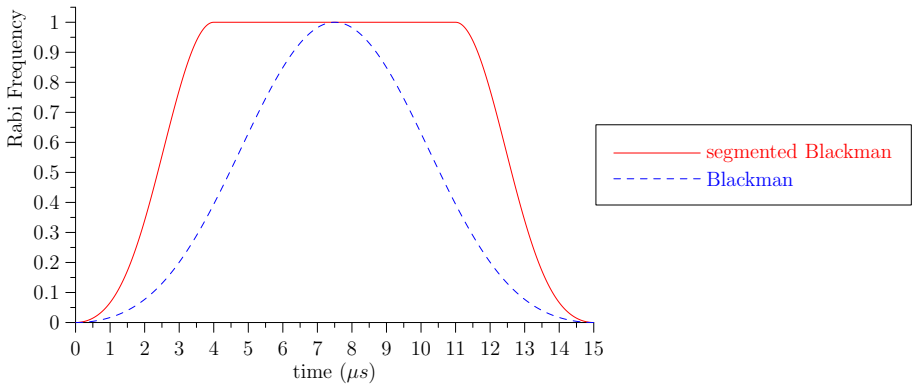

Figure 12: The segmented Blackman pulse and the full Blackman pulse in comparison. At the same peak Rabi frequency the pulse area of the segmented Blackman pulse is larger.

same peak Rabi frequency and pulse duration. This means that the relative pulse area $A_{rel} = 1/(\Omega_{max} t_{pulse}) \int_{t=0}^{t=t_{pulse}} \Omega(t)\, dt$ has to be close to one. The relative pulse area of the Blackman function as defined above is $A_{rel} = 0.42$. Using this function as an amplitude shape would double the time required for a full Rabi oscillation as compared to a rectangular pulse. Our solution to this problem is to split the pulse into a rising slope, a constant plateau and a falling slope. Such a segmented pulse is compared to a normal Blackman pulse in Fig. 12. For durations of the rising and falling slopes of each 5% of the total pulse duration, this leads to a relative pulse area of $A_{rel,total} = 0.9 + 0.1 \cdot A_{rel,slope}$. This approach has the additional advantage that for different pulse lengths the same rising and falling slopes can be used which simplifies the generation of the pulses. Using such a segmented pulse as a windowing function leads to larger off-resonant excitations than using the full windowing function, but the smooth slope of the windowing functions is preserved. In terms of the Fourier transform this window may be seen as the convolution of a short Blackman window and a rectangular window, which shows properties of both the Blackman and a rectangular window. To assess the quality of a pulse shape, different pulse shapes are compared in this section. The different shapes are defined in Tab. 3. Fig. 13 shows rising slopes of cosine, linear and Blackman shaped pulses.

Name	Expression
cosine	$\Omega(t) = \Omega_o \frac{1}{2}(1 - \cos(\frac{\pi t}{T}))$
linear	$\Omega(t) = \Omega_o \frac{t}{T}$
Blackman	$\Omega(t) = \Omega_0 \frac{1}{2}\left(0.84 - \cos(\frac{t\pi}{T}) + 0.16 \cos\frac{2\pi t}{T}\right)$

Table 3: The three different slope functions used in the simulations. A pulse consists of a rising slope with the slope duration $t_{slope} = T$, a constant plateau where $\Omega(t) = \Omega_0$ and a falling slope which is symmetric with respect to the rising slope.

The effect of the pulse shape on adiabaticity In this section a method to estimate whether a pulse shape is in the adiabatic regime without calculating the whole time evolution is presented. Since the condition for applying the adiabatic approximation is $|\dot{\Omega}(t)\,\delta| \ll |\tilde{\Omega}(t)|^3$, a measure of the adiabaticity of a shaped laser pulse is given by the

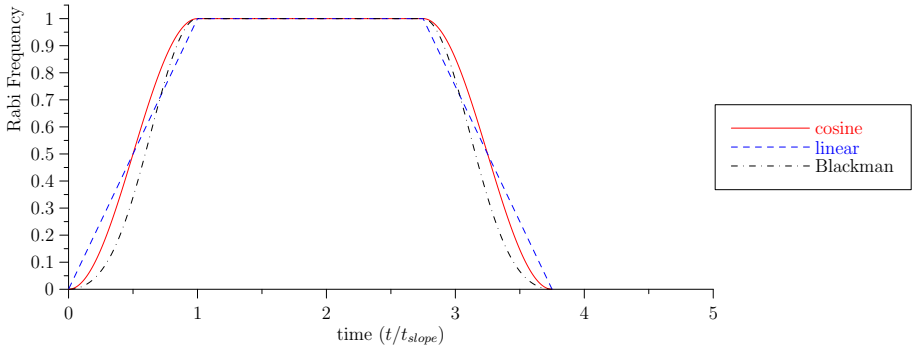

Figure 13: The rising slopes of the cosine, linear and the Blackman pulse shapes as defined in Tab. 3.

coefficient of adiabaticity:

$$\alpha = |\dot{\Omega}(t)\,\delta|/|\tilde{\Omega}(t)|^3$$

The criterion for adiabaticity is fulfilled if $\alpha \ll 1$. For the pulse shapes presented above, the coefficient of adiabaticity is determined by the Rabi frequency Ω, the detuning δ and the slope duration t_{slope}. As the Rabi frequency is not constant in time, α also varies with time. Therefore α_{max}, the maximum value of α over a whole pulse may be used to estimate the off-resonant excitation of the pulse shape. There are two different interesting regimes for α:

- The transition regime where $\delta \approx \Omega$.
- The adiabatic regime where $\delta^2 \gg \Omega^2$.

Driving a sideband transition with our usual experimental parameters ($\Omega = 2\pi\,200$kHz, $\delta \approx 1$MHz) leads to $\delta^2/\Omega^2 = 25$ which implies that the system is in the adiabatic regime. However, the off-resonant excitations are too small to be measured directly with these parameters. Direct measurement of off-resonant excitations is only possible in the transition regime. In Fig. 14 the time dependence of α during a rising slope is shown for the two regimes.

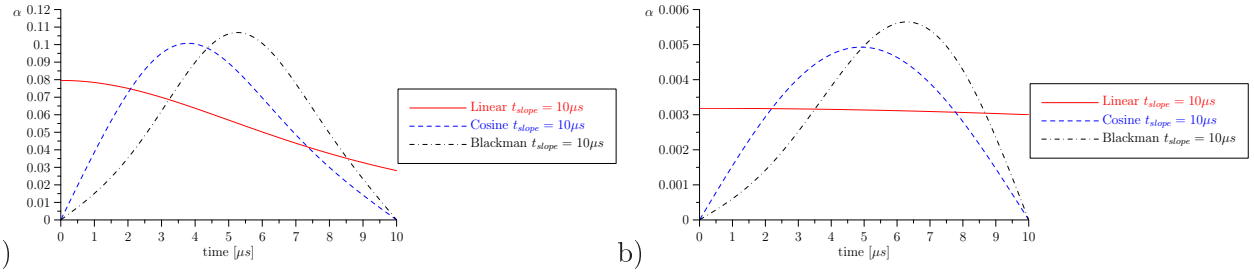

Figure 14: Time dependence of the adiabaticity factor α of different shapes. The Rabi frequency and the slope durations are in both cases $\Omega = 2\pi\,200\,\text{kHz}$ and $t_{slope} = 10\mu$s. The detuning is a) $\delta = 2\pi\,200\,\text{kHz}$ and b) $\delta = 2\pi\,1\,\text{MHz}$.

For the system being in the adiabatic regime it is possible to perform a second order Taylor expansion of α to achieve a simple analytical result for α:

$$\alpha = \frac{\dot{\Omega}\delta}{(\Omega^2+\delta^2)^{3/2}} \approx \frac{\dot{\Omega}}{\delta^2}(1-\frac{3}{2}\frac{\Omega^2}{\delta^2}) \approx \frac{\dot{\Omega}}{\delta^2}$$

It is therefore possible to derive simple analytical results for α_{max} if $\delta^2 \gg \Omega^2$:

$$\begin{aligned} \alpha_{max,cos} &\approx \frac{\pi}{2}\frac{\Omega_0}{\delta^2\, t_{slope}} \\ \alpha_{max,lin} &\approx \frac{\Omega_0}{\delta^2\, t_{slope}} \\ \alpha_{max,black} &\approx \frac{1.15\,\pi}{2}\frac{\Omega_0}{\delta^2\, t_{slope}} \end{aligned}$$

Usual experimental parameters for driving a sideband transition are a detuning of approximately $\delta = 2\pi\,1\,\mathrm{MHz}$ and a Rabi frequency of $\Omega = 2\pi\,200\,\mathrm{kHz}$. Using these parameters and a slope duration of 5μs leads to the following values of α_{max} :

$$\begin{aligned} \alpha_{max,cos} &\approx 0.010 \\ \alpha_{max,lin} &\approx 0.006 \\ \alpha_{max,black} &\approx 0.012 \end{aligned}$$

In Fig. 15 α_{max} is shown for varying detuning for $\Omega_0 = 2\pi\,463\,\mathrm{kHz}$ and a slope duration of $t_{slope} = 10\,\mu$s.

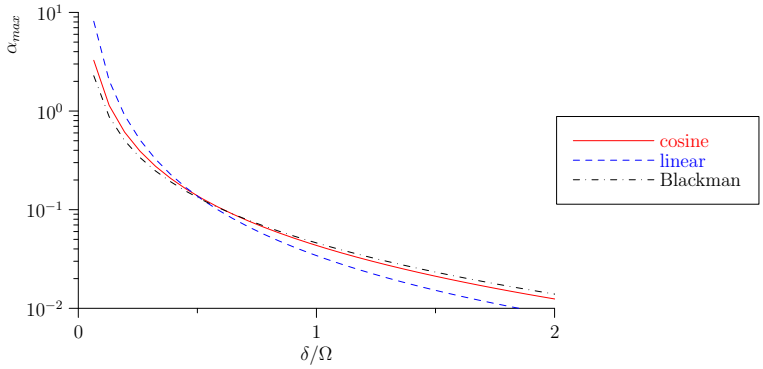

Figure 15: The maximum of the adiabaticity factor α_{max} for different pulse shapes as a function of the detuning for a Rabi frequency of $\Omega_0 = 2\pi\,463\,\mathrm{kHz}$ and a ramp duration of $T = 10\,\mu$s .

2.3 Numerical calculations

Numerical simulations are performed with the following algorithm. Time is divided into discrete steps where the Rabi frequency is assumed to be constant for the duration of one time step. The state at the end of the time step is calculated by transforming the previous state from an unperturbed basis into the dressed basis, calculating the phase evolution and then transforming the state back into the unperturbed basis. Eq. 13 describes

the transforming operation. The resulting state is then used as an input state for the next iteration. The Matlab script used for calculating the time evolution is shown in appendix(D). If not stated otherwise the Rabi frequency used for the results presented below is $\omega_0 = 2\pi\,463\,\mathrm{kHz}$. This value is chosen to match the experimental data which were first taken at this Rabi frequency.

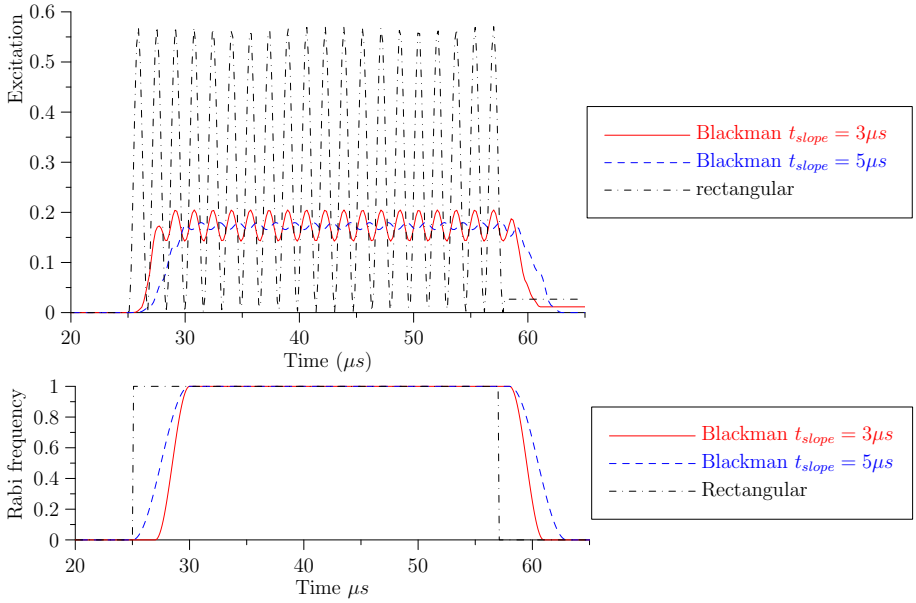

Figure 16: Simulated time evolution for the system initially in the ground state for a Rabi frequency of $\Omega_0 = 2\pi\,463\,\mathrm{kHz}$ and a detuning of $\delta = 2\pi\,400\,\mathrm{kHz}$

Typical time evolutions of the populations for some pulse shapes are shown in Fig. 16. The remaining off-resonant excitations are quantified by simulating the system multiple times with different pulse lengths. From each simulation only the excitation at the end of the pulse is used for further analyzation. This excitation oscillates with varying pulse length. A squared sine function is fitted to these oscillations and its amplitude is the magnitude of the off-resonant excitations.

As stated in section(2.1.5) the system is in a dressed state during the entire pulse if the pulse is adiabatic and the system is initially in the state $\Psi_0 = \Psi_1$. Therefore no population oscillations should occur for an adiabatic pulse shape. The time evolutions of the Blackman pulses in Fig. 16 show that the oscillations vanish with increasing slope duration which is an successive approximation to the adiabatic behaviour.

To compare the different shapes, the remaining off-resonant excitations are analyzed for different parameters. In Fig. 17 the off-resonant excitations of the cosine, linear, Blackman and rectangular shapes are shown as a function of the detuning δ. The Blackman and the cosine shapes behave nearly identically. It can also be seen that for small detunings the linear shape suppresses off-resonant excitations slightly better than the cosine shape. Nevertheless, the cosine and Blackman shaped pulses suppress residual off-resonant excitations better for large detunings which corresponds to the typical situation our experiment. Another possibility for characterizing the pulse shape is to look at off-resonant excitations

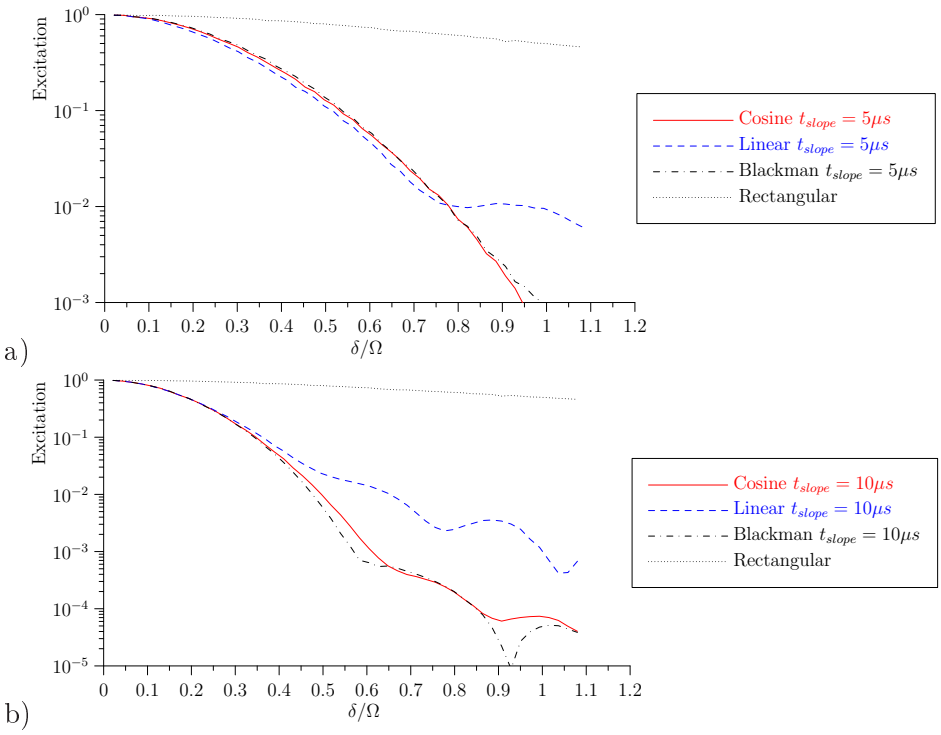

Figure 17: Simulated off-resonant excitations as a function of the detuning for a Rabi frequency of $\Omega_0 = 2\pi\,461\,\text{kHz}$. The slope durations are at a) $t_{slope} = 5\mu$s and at b) $t_{slope} = 10\mu$s

as a function of the slope duration. The results of these calculations are shown in Fig. 18. As expected, the linear shape behaves more favorable at shorter slope durations than the cosine and the Blackman pulse shapes and vice versa.

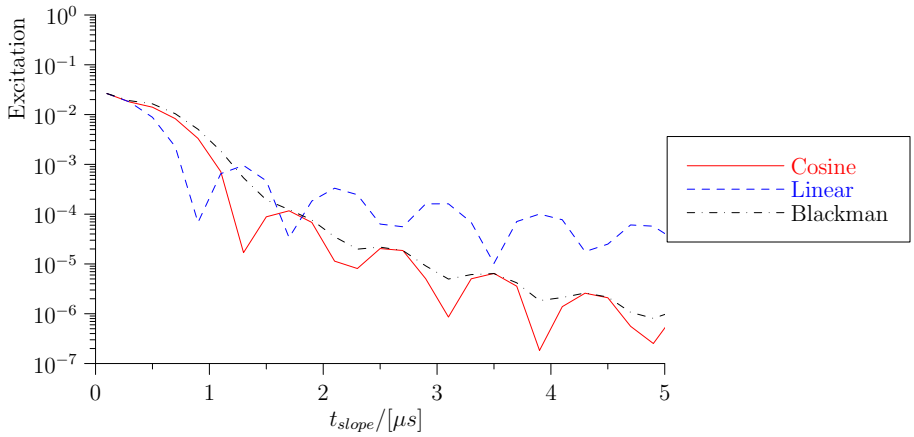

Figure 18: Simulated dependence of off-resonant excitations on the ramp duration. The parameters are $\Omega_0 = 2\pi\,463\,\text{kHz}$ and $\delta = 2\pi\,340\,\text{kHz}$.

Previously the coefficient of adiabaticity α was introduced. To test whether this coefficient provides a good characterization for off-resonant excitations the results of the numerical simulations are plotted as a function of α.

In Fig. 19 the dependence of α is shown for a cosine pulse shape for the regimes as defined above. In Fig. 20 the same plot is shown for all three pulse shapes. It can be seen

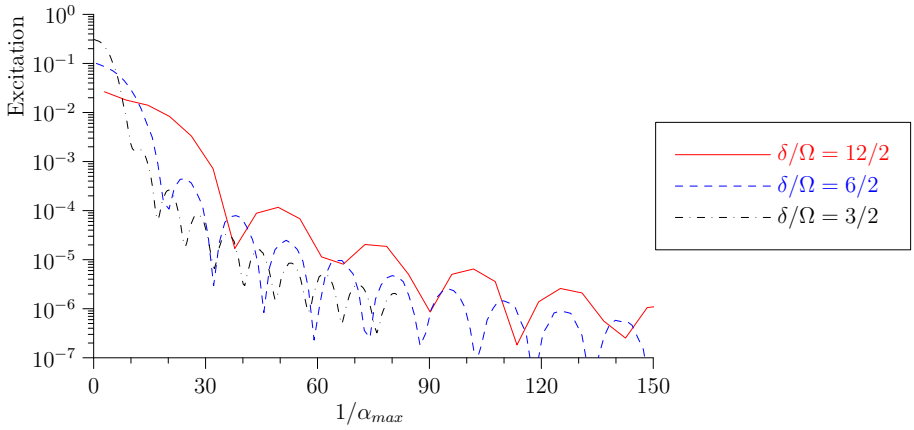

Figure 19: Simulated off-resonant excitations for cosine pulse shapes as a function of the inverse of the maximum adiabatic factor $1/\alpha_{max}$.

that even for small α the off-resonant excitations differ for the different pulse shapes. This is due to the fact that α_{max} does not resemble the exact deviation from the ideal adiabatic time evolution. The linear shape scales less favorable with α than the cosine and Blackman shapes. This is because α does not vanish at the beginning and the end of the slope for the linear shape. Furthermore it can be seen that to achieve off-resonant excitations of less than 1% it is sufficient to chose the slope duration such that $\alpha < 1/40$.

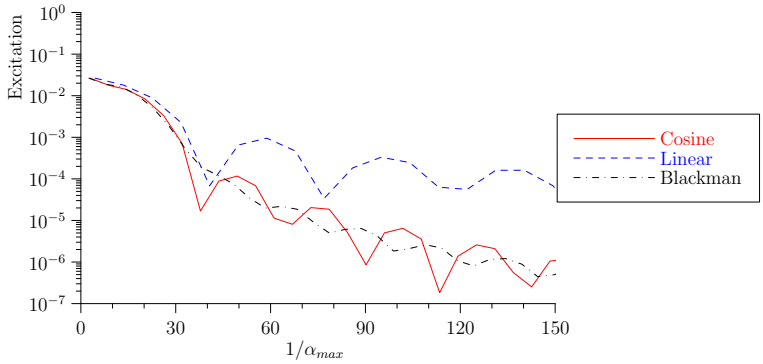

Figure 20: Simulated off resonant excitations for different pulse shapes as a function of the inverse of the maximum adiabatic factor $1/\alpha_{max}$.

The simulated remaining off-resonant excitations when driving a sideband transition with a Blackman shaped pulse with a slope duration of 3μs, an axial trap frequency of 1MHz and a Rabi frequency of $\Omega = 200$kHz are $P_D = 3.5 \cdot 10^{-5}$. For a linear shaped pulse with the same parameters the off-resonant excitations have a magnitude of $P_D = 10 \cdot 10^{-5}$.

Simulation for different initial states

As stated in section(2.1), off-resonant excitations change their qualitative behaviour if the system is not initially in the ground state. The most general single qubit state is $\Psi_0 = \gamma_1 \Psi_1 + \gamma_2 \Psi_2$ where $\gamma_{1,2}$ may be complex valued and the normalization condition has to be fulfilled. Due to the possibility of complex values for $\gamma_{1,2}$ it is not easily possible to

cover this whole space with numerical calculations. To be able to perform simulations in a reasonable time, only real values of $\gamma_{1,2}$ are used for the simulations. This is sufficient as the only difference is a constant phase offset which has no effect on off-resonant excitations. The corresponding parameter is the mixing angle Θ which is defined by $\gamma_1 = \sin\Theta$ which leads to $\gamma_2 = \sqrt{1-\gamma_1^2}$.

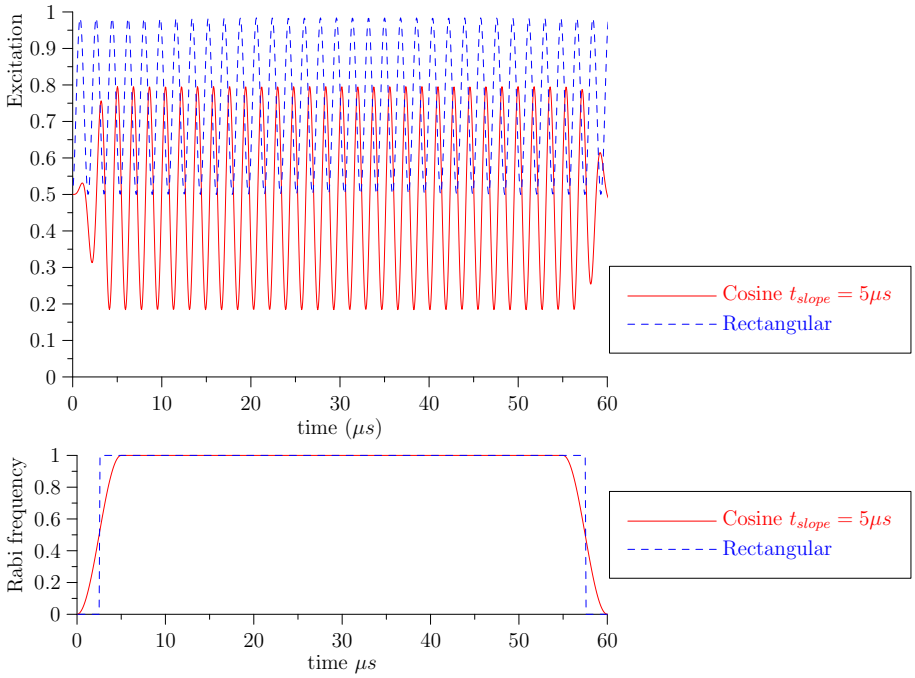

Figure 21: Simulated time evolution for initial state $\Psi = 1/\sqrt{2}(|0\rangle + |1\rangle)$.

In Fig. 21 the time evolution for the initial state $\Psi = 1/\sqrt{2}(|0\rangle + |1\rangle)$ is shown for a rectangular and a cosine shaped pulse with $5\mu s$ slope duration. Generally the initial state is not a dressed state of the system and therefore the population oscillations do not vanish during the pulse. Nevertheless, according to the calculations in section(2.2) the off-resonant excitations vanish at the end of the pulse if the interaction strength is switched on and off adiabatically.

In Fig. 22 the dependence of the remaining excitations on the mixing angle Θ is shown. As shown in section(2.1.7) the excitations in dependence of the mixing angle θ are $P_{exc} = \cos^2(\theta) - \sin(2\theta)\,\epsilon + \left(1 - 2\cos^2(\theta)\right)\epsilon^2$ where ϵ is the error angle on the qubit. Neglecting the terms with ϵ^2 it is possible to derive an expression for the error which is independent of the mixing angle. The error is expressed in ϵ^2 to preserve consistency with the results given for the initial state being the ground state. The equation for calculating ϵ^2 from the amplitude of the population excitations is then $\epsilon^2 = A_{pop}/\left(4\,\sin^2(2\theta)\right)$. One has to bear in mind that this result is only valid for $\sin(2\theta) \gg \epsilon$. In Fig. 23 the calculated ϵ^2 is shown in dependence of the mixing angle θ. It can be seen that the error stays almost constant for all mixing angle. From this we infer that the chosen measure for off-resonant excitations is suitable for an arbitrary input state.

From Fig. 22 and Fig. 23 it seems that the cosine pulse shapes behaves more favorable

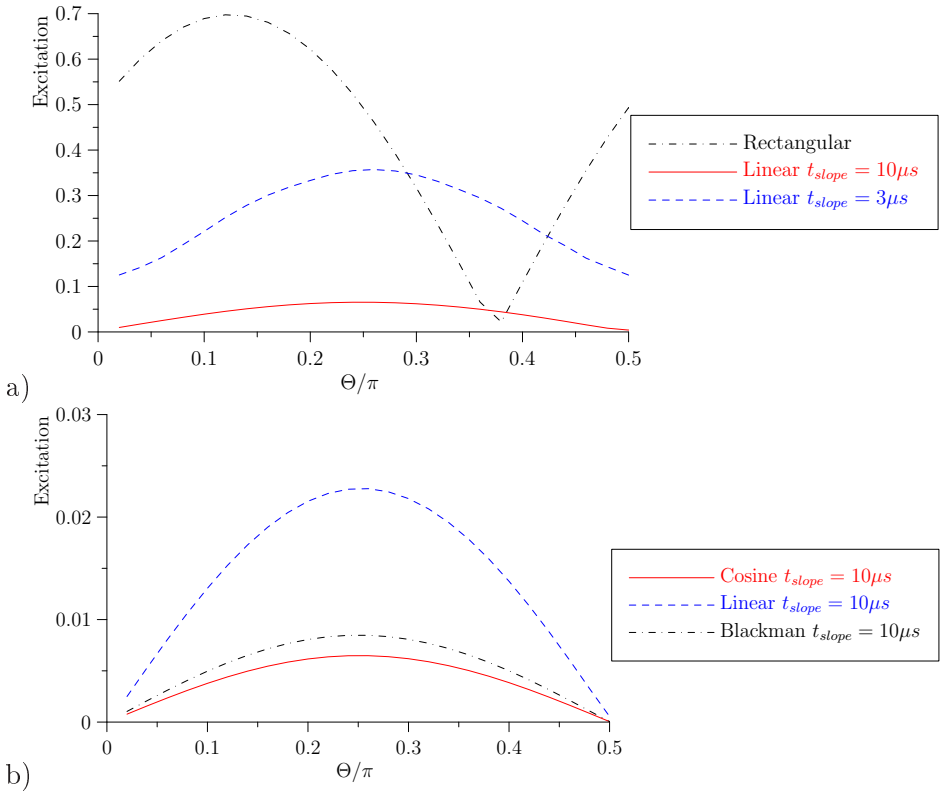

Figure 22: Simulation for different initial states for different pulse shapes. The initial state is $\Psi = \sin\theta\ |\Psi_2\rangle + \cos\theta\ |\Psi_1\rangle$. a) compares a rectangular and linear shapes. b) shows different shapes with $t_{slope} = 10\mu$s

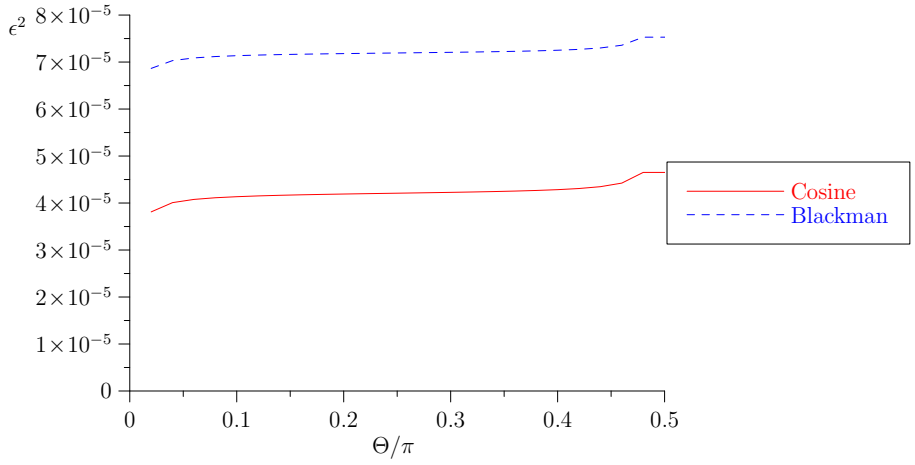

Figure 23: Calculated error angle ϵ^2 for simulated data as a function of the mixing angle θ. The parameters for the simulations were $\Omega = 2\pi\,200\,\text{kHz}$, $\delta = 2\pi\,400\,\text{kHz}$, $t_{slope} = 10\mu$s.

than the Blackman shape if the initial state differs from the ground. In Fig. 24 the error angle ϵ^2 is shown in dependence of the slope duration for two different initial states . It can be seen that the cosine shape behaves more favorable for certain parameters, but the general dependence is similar for cosine and Blackman shaped pulses.

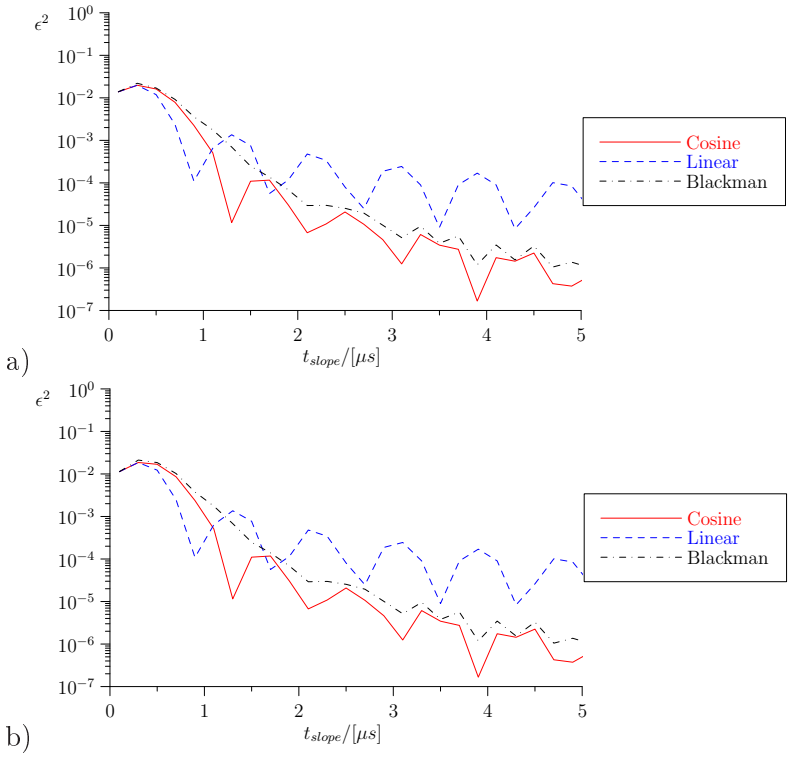

Figure 24: Calculated error angle ϵ^2 for simulated data as a function of the slope duration for initial state of a) $\theta = \frac{1}{4}\pi$ and b)$\frac{2}{3}\pi$. The parameters of the simulations were $\delta = 2\pi\,1\text{MHz}$ and $\Omega = 200\text{kHz}$.

2.4 The AC Stark effect

When a transition is driven off-resonantly, population oscillations are not the only undesirable side effect. The phase of the qubit evolves linearly with the pulse duration. In section (2.1.5) the solution for an adiabatic pulse was derived:

$$A_{\pm}(t) \approx \exp\left(\pm\frac{i}{2}\int_0^t dt'\,\tilde{\Omega}(t')\right) A_{\pm}(0)\ .$$

As the dressed basis is the original basis at the beginning and the end of the pulse this equation may be rewritten as

$$C_{1,2}(t_{end}) \approx \exp\left(\pm\frac{i}{2}\int_0^t dt'\,\tilde{\Omega}(t')\right) C_{1,2}(0)\ .$$

To investigate the phase evolution of this state the integral has to be evaluated for varying pulse lengths. The laser pulse is divided into three sections and therefore also the integral can be split. As the rising and the falling slopes are the same for different pulse lengths these lead only to a constant factor C_{slope}. Furthermore the Rabi frequency is constant in the third part and therefore the resulting phase evolves linearly with increasing pulse length. With the pulse length being $t_{end} = t_{top} + 2t_{slope}$ this leads to

$$C_{1,2}(t_{end}) = \exp(C_{slope} + \tilde{\Omega}t_{top})\, C_{1,2}(0)\ .$$

One has to bear in mind that the phase of the qubit is defined in the frame of reference which rotates with the transition frequency ω_0. However, the calculations above have been made in a frame rotating with the laser frequency $\omega = \omega_0 + \delta$. In this frame the state vector oscillates with the effective Rabi frequency $\tilde{\Omega}$ when applying an off-resonant pulse. This leads to the following oscillation frequency in the qubit frame:

$$\Omega_{qubit} = \tilde{\Omega} - \delta = \sqrt{\delta^2 + \Omega^2} - \delta$$

For large detunings ($\delta^2 \gg \Omega^2$) a first order Taylor expansion of $\tilde{\Omega}$ can be used to estimate the effect:

$$\Omega_{qubit} = \delta\sqrt{1 + \frac{\Omega^2}{\delta^2}} - \delta \approx \frac{\Omega^2}{2\delta}$$

The physical reason for this dephasing are frequency shifts on the qubit transitions which are known as the AC Stark shift. To investigate the Stark shift for our qubit, the Zeeman structure of the $S_{1/2} \rightarrow D_{5/2}$ transition and the dipole interactions have to be included in the calculation. There is no need to account for the sideband transitions, as the coupling strength is much weaker. The detuning for the dipole interactions ($S_{1/2} \rightarrow P_{1/2}$, $S_{1/2} \rightarrow P_{3/2}$ and $D_{5/2} \rightarrow P_{3/2}$) is in the range of several hundred THz for a laser which is close to resonance with the quadrupole transition. The Rabi frequency of the dipole transition is about a factor of thousand bigger than the Rabi frequency of the quadrupole transition for the same light field. As the Stark shift is inversely proportional to the

detuning but quadratic with the Rabi frequency, the contributions of the dipole transitions cannot be neglected with a typical detuning of 1MHz from the quadrupole transition. Each level is shifted by

$$\Delta_i = \pm\frac{\Omega_i^2}{4\delta_i}$$

where Ω_i is the Rabi frequency of the applied laser field and δ_i the detuning for the particular transition. The sign of the shift depends on whether the applied light is red or blue detuned. The level shifts for the Zeeman sublevels of the allowed transitions for our geometric configuration from the $S_{1/2}$ to the $D_{3/2}$ are shown in Fig. 25.

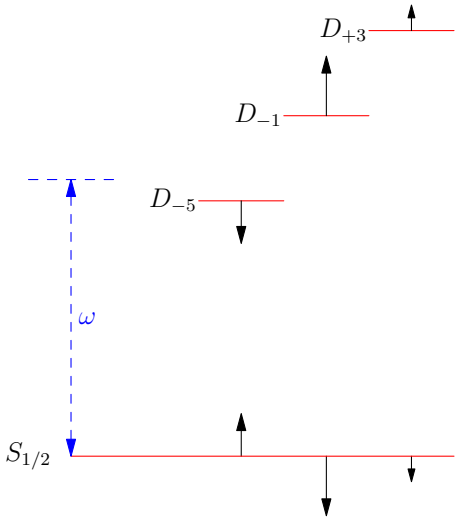

Figure 25: AC-stark shifts for the allowed transitions from the $S_{1/2}(m = -1/2)$ to the $D_{3/2}$ level. D_{-5} refers to the $D_{3/2}(m = -5/2)$ state. The shifts are shown for a blue detuned laser light driving a motional sideband.

To calculate the total Stark shift induced by a laser pulse, the contributions of the different atomic levels are added [19, 20]. This leads to the equation $\Delta_{AC} = \Omega^2/4 \sum_i a_i/\delta_i$ for the Stark effect generated by the Zeeman sublevels. The factors a_i are the coupling strengths of the different Zeeman components relative to the qubit transition and Ω is the Rabi frequency of the qubit transition. As the dipole levels are generally far detuned with respect to the light field, they may be included in this calculation by a term which does not depend on the detuning from the quadrupole transition. The total Stark shift is then

$$\Delta_{AC} = \frac{\Omega^2}{4}\left(\sum_i \frac{a_i}{\delta_i} + b\right)$$

where the factor b is the sum of the contributions of the dipole interactions to the Stark shift. In reference [20] this shift was measured and it was shown that the resulting Stark shift is negative when applying a blue sideband pulse. Therefore it is possible to compensate this shift with a second far detuned laser which causes a positive light shift on the dipole transition but only a negligible shift on the quadrupole transition [16, 19, 20]. The amplitude of this compensation pulse is adjusted to minimize the AC Stark effect

and it must have the same length as the manipulating pulse. Since the acousto-optical modulator that generates the laser pulses operates in a linear regime it is straightforward to extend this scheme to shaped laser pulses. If the compensation pulse has the same pulse shape as the manipulation pulse and appropriate peak laser intensity, the Stark shift is compensated at every time during the pulse.

3 Experimental setup

3.1 The ion trap

The ion trap used in our experiment was designed by Stephan Gulde and is discussed in detail in his PhD thesis [19]. The ion trap is operated with a frequency of approximately 23 MHz. The radio frequency voltages required to create a radial trapping potential are generated with a high power radio frequency amplifier and a resonant circuit. The DC end cap voltage required for axial trapping is generated by a digitally controlled multi-channel power supply[2]. The trap is mainly operated with RF powers of around 10W and end-cap voltages of 1000V which corresponds to a center-of-mass mode frequency in the axial direction of about 1.2MHz. The inter-ion distance depends on the end-cap voltage and therefore the parameters of the trap are adjusted to optimize addressing errors and off-resonant excitations for a given number of ions.

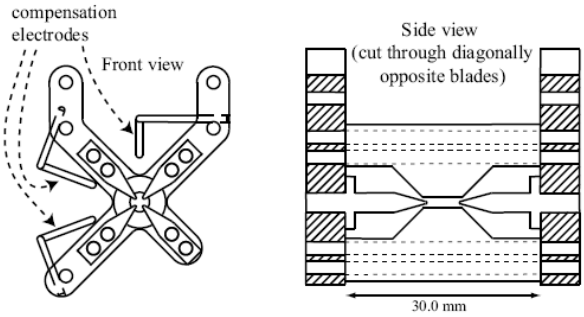

Figure 26: Schematic drawing of the ion trap used in our experiment. The compensation electrodes are used to minimize micro-motion.

3.2 Optical setup

The optical setup is distributed over three optical tables: two laser tables which are used simultaneously by all $^{40}Ca^{+}$ experiments, and an experiment table where the vacuum apparatus is located. In this thesis only a short overview of the optical setup is given, for more details see Mark Riebe's or Stephan Gulde's PhD thesis [16, 19].

3.2.1 Lasers

For each transition described in chapter 1.2 a separate laser is used. Tab. 4 shows the types of lasers used. All diode lasers are Toptica DL 100, where the 854nm and 866nm lasers are frequency stabilized to a temperature stabilized cavity via the Pound-Drever-Hall method [21]. The Titanium-Sapphire lasers are Coherent 899 ring lasers, which are optically pumped by Coherent Verdi lasers. The 397nm light for driving the detection transition is generated via the frequency doubling stage which has as input 794nm light coming from a Titanium-Sapphire laser. The 729nm laser is locked to a very high finesse ($f \approx 500\,000$) cavity.

[2] ISEG EHQF2020p

Wavelength	Usage	Laser type
729 nm	Qubit transition / Doppler Cooling	Titanium Sapphire
397 nm	Detection / Doppler cooling / Optical pumping	Titanium Sapphire
866 nm	Detection / Repumping	Diode
854 nm	Sideband cooling repumping / State initialization	Diode
422 nm	Photoionization	Diode
375 nm	Photoionization	Diode

Table 4: The lasers used in the experiment. The 866nm , 854nm and 397nm lasers have a linewidth of around 100kHz. The linewidth of the 729nm laser is around 50Hz. The photoionization lasers are not stabilized.

The 729nm setup A prerequisite for high fidelity qubit operations and a long qubit coherence time are small intensity and phase fluctuations of the 729nm laser. The linewidth of this laser was measured to be 48Hz on a timescale of one minute [22], which corresponds to a decoherence time of about 10ms [23]. Beat measurements with an identically set-up laser at the $^{43}Ca^{+}$ experiment have shown a combined linewidth of less than 5Hz in 1s. At the ions position, the laser is focused to a spot size smaller than the inter-ion distance and it is possible to switch between different ions during an experimental cycle.

Since the laser is placed on a different optical table than the vacuum vessel, the light is transferred to the experiment table via a mono-mode fiber. Intensity and phase noise fluctuations induced by the fiber are compensated with an active phase and intensity stabilization circuit [24]. The light is then coupled into a double pass acousto-optical modulator (AOM). In this AOM the actual frequency and amplitude of the laser manipulating the ions is determined. The light is then coupled into another fiber going into a closed box where the beam is focused and deflected to achieve single ion addressing. To achieve this, the beam passes a telescope with one lens mounted on a translation stage, the electro optical deflector (EOD) and a dichroic mirror which reflects 729nm light and transmits 397nm light. In front of the vacuum chamber is a microscope lens[3] which focuses the beam to a width of about $2\mu m$ which is smaller than the typical inter-ion distance of about $5\mu m$. The path backwards through the objective is used to detect 397nm light with an EMCCD camera[4]. A simplified schematic for the optical setup for ion addressing is shown in Fig. 27.

The infrared laser setup The 866nm and 854nm lasers are placed on a separate optical table. Their light is transmitted to the experiment table with one mono-mode fibre. The lasers used are Toptica DL100 diode lasers stabilized to a reference cavity with the Pound Drever Hall method. The linewidth of the locked laser is in the 100kHz regime, which is much smaller than the linewidth of the used transitions. On the experiment table the infrared lasers and the 397 laser used for Doppler cooling and detection are coupled into the same photonic crystal fibre which is mono-mode for both wavelengths.

[3] Nikon MNH-23150 ED Plan 1.5x
[4] Andor iXon DU860AC-BV

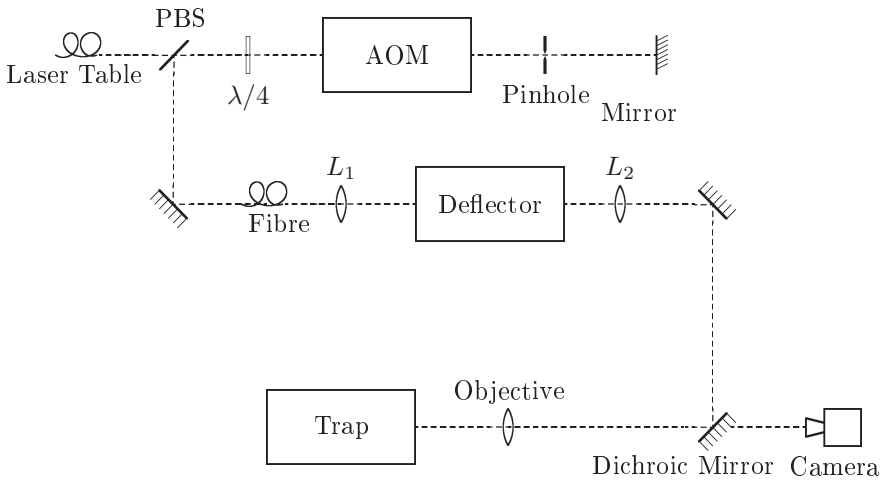

Figure 27: Simplified 729nm laser beam path.

The blue laser setup The 397nm light for optical pumping, detection and Doppler cooling is generated by a frequency doubled Titanium-Sapphire laser which is stabilized to a cavity in the same vacuum vessel as the infrared lasers. The laser is also transmitted to the experiment table with a mono-mode fibre. After the fibre the laser is split into two beams, one used for Doppler cooling and detection, and another beam for initial state preparation. The second beam is aligned along the trap axis and has circular polarization. Due to its polarization this beam is referred as the "sigma" beam. The detection path is fed through an acousto optical modulator and then coupled in the photonic crystal fibre together with the infrared light.

3.2.2 The detection system

State detection can be carried out using two different methods, using either a photo multiplier tube (PMT) or an intensified CCD camera. The PMT is used for state detection when loading the trap and for experiments where pulses are generated conditionally depending on the outcome of a measurement. The CCD camera can be used when state detection of multiple ions is required.

The PMT and the camera are mounted along the axis on which the 729nm light is shone into the trap. Stray light is suppressed by using optical filters in front of the detection devices. The camera is located behind the microscope objective which is used to focus the 729nm beam. The 729nm and the 397nm beam are separated by a dichroic mirror. The camera image is evaluated on a dedicated computer. For more information on state detection see Mark Riebe's PhD thesis [16].

3.3 Experiment control

In this section the experiment control setup of the $^{40}Ca^{+}$ experiment is described. The most important parts of the experiment are directly controlled by the experiment control computer using a LabView control program. The most notable computer controllable

experimental parameters are:

- Frequency and intensity of the 397nm, 866nm and 854nm lasers via VCOs[5] and variable attenuators, which are controlled by analog voltages.
- End-cap voltages, which are generated by a digitally controlled high voltage supply.
- Position of the lens in the 729nm addressing box via a digitally controlled XYZ translation stage.
- Voltage on the electro optical deflector in the addressing box generated by an analog voltage and a high voltage amplifier.
- Frequency and Intensity of the 729nm laser via direct digital synthesis and a variable gain amplifier.

As mentioned in section(1.2), an experimental cycle consists of an exactly timed sequence which is repeated around 100 times. The experimental controls are therefore divided into synchronous and asynchronous controls. The synchronous controls need to be changed within one of these experimental cycles whereas it is sufficient to change asynchronous controls between different sequences. The most challenging control parameter is the analog radio frequency signal driving the AOM for coherent manipulation of the qubit. The two different setups are named with respect to the method of their radio frequency generation.

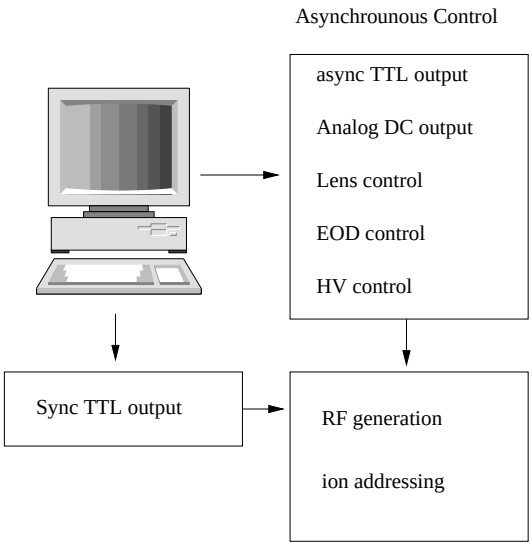

Figure 28: The Marconi setup. The synchronous control signals are generated by a DIO64 timer card. The asynchronous analog and digital signals are generated by a National Instruments NI6703 card.

The Marconi setup An overview of the Marconi setup is shown in Fig. 28. The synchronous controls are generated via a timer card from National Instruments[6].This card is

[5] Voltage controlled oscillator
[6] NI DIO64

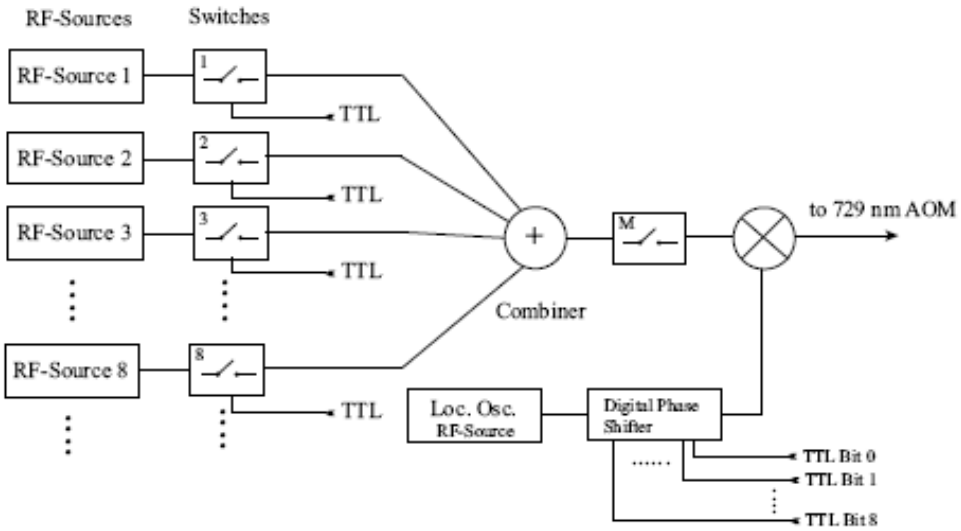

Figure 29: Previously used 729nm radio frequency setup. Each RF source shown is realized by an individual Marconi Synthesizer. (Image from [16])

is able to generate exactly timed digital TTL pulses. It may be triggered by an external input to synchronize the experiment with the mains line phase. As stated above the most demanding part of the experiment control is the creation of the radio frequency signals for qubit manipulation. The required signals are generated by radio frequency synthesizers[7]. These synthesizers are not controllable within a sequence and therefore have to be programmed prior to each sequence and switched with the help of digitally controlled radio frequency switches[8]. As multiple frequencies are required within one sequence, the set-up grew to consist of nine frequency synthesizers connected in a complicated switch network which is shown in Fig. 29. Analog and digital channels which don't change during a sequence were controlled asynchronously by a NI6703 output card. The only analog channel which needs to be changed synchronously during a sequence is the voltage of the EOM required for ion addressing. This is achieved by writing the sequence of required voltages in advance into the buffer of the fast NI6713 output card. The card is then triggered by synchronous digital pulses.

The programmable pulse generator setup Fig. 30 shows the currently used setup, which uses the programmable pulse generator (PPG) to generate all synchronous signals. The programmable pulse generator is able to generate exactly timed digital and radio frequency pulses. It is able to generate radio frequency signals with different frequencies and phases within one sequence. It was originally designed by Paul Pham and is described in more detail in the section below. The generation of the asynchronous control signals is unchanged with respect to the Marconi setup. The ion addressing scheme is generally the same as

[7] Marconi 2032A
[8] Mini Circuits ZYSW-2-50DR

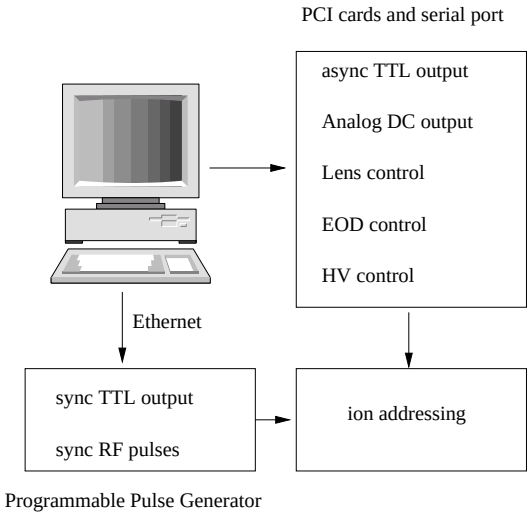

Figure 30: The programmable pulse generator electronic setup. Asynchronous analog and digital control voltages are controlled via a National Instruments PCI card. The synchronous digital and radio frequency outputs are controlled by the programmable pulse generator which is connected to the experiment control computer via a standard Ethernet interface.

for the Marconi setup, however the digital trigger is now generated by the programmable pulse generator.

3.4 The programmable pulse generator

3.4.1 Overview

As described in the section above, the experimental sequences are generated by the programmable pulse generator (PPG). The programmable pulse generator is a device which is designed to generate exactly timed digital (TTL) and radio frequency (RF) pulses. As shown in Fig. 31 the programmable pulse generator may be subdivided in four major blocks:

- Communication via Ethernet with the experiment control computer
- Timing control and program flow control
- Radio frequency pulse generation
- Digital output system

The heart of the programmable pulse generator is a complex programmable logic chip (field programmable gate array). A field programmable gate array (FPGA) is a reconfigurable logic device which consists of small logic blocks capable of carrying out arbitrary logic functions. The FPGA used in the programmable pulse generator[9] has over 12 000 logical units, and therefore it is possible to realize complex designs (for example an entire microprocessor). The programming of the FPGA is accomplished in a hardware

[9] Altera Cyclone EP1C12

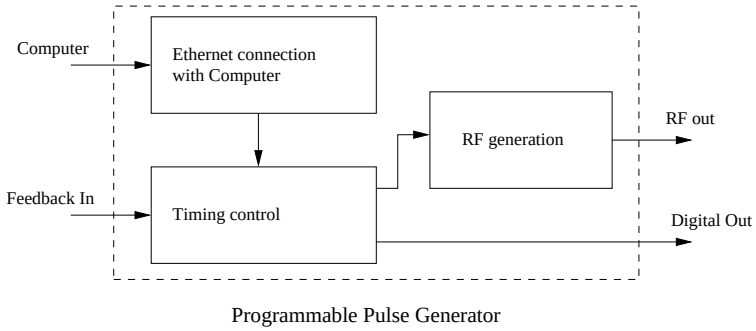

Figure 31: Logical block diagram of the programmable pulse generator. The communication, timing control, digital output subsystem is located in a programmable logic chip. The radio frequency is generated by an external synthesizer.

description language (HDL). The VHSIC[10] hardware description language (VHDL) is used in this project. The advantage of using a standard hardware description language is the ability to generate a design independent of the actual type of hardware and independent of the development environment and ensuring therefore portability. The steps for generating a design in an FPGA are:

- Hardware description (VHDL)
- Behavioral Simulation
- Synthesis and Optimization
- Timing Simulation
- Device programming

For the complete design flow the freely available web edition of the Quartus II design suite from the FPGA manufacturer Altera[11] is used.

The hardware block diagram for the programmable pulse generator is shown in Fig. 32. All logical blocks except the radio frequency generation are integrated in the FPGA. The hardware for the FPGA was designed by Paul Pham and is described in his Master's thesis [25]. The communication with the computer is realized over a standard Ethernet interface via a custom protocol. Therefore no additional hardware on the experiment control computer is required. A more detailed overview of the programming of the FPGA is given in section(3.4.3).

The FPGA has only digital outputs and therefore the radio frequency pulses are generated by a direct digital synthesizer (DDS). Direct digital synthesizing is a technique of generating an analog radio frequency output from a stable digital clock. Due to its digital nature the direct digital synthesizer offers better control over the generated output than analog techniques to generate radio frequency signals (VCOs, PLL, etc). This makes phase coherent frequency switching within one sequence with only one synthesizer possible.

[10] Very-High-Speed Integrated Circuits
[11] http://www.altera.com

For a more detailed description of direct digital synthesizer operation see section(3.4.2). The programmable pulse generator adds the ability to generate amplitude modulated radio frequency pulses. This is realized with a digital to analog converter (DAC) in combination with a variable gain amplifier (VGA). The digital to analog converter and the direct digital synthesizer are controlled by the digital outputs of the FPGA. With the help of additional addressing electronics, the FPGA is able to control up to 16 synthesizers and 16 digital to analog converters. A general purpose I2C (Inter-Integrated Circuit) serial bus is also implemented in the FPGA programming. This bus may be used to interface different microprocessors and measurement devices, but so far use of this bus has not been required.

Since several research groups are using the programmable pulse generator, it is managed as an open source project on the Sourceforge Project management homepage[12]. The software is released under the BSD open source license[13]. The most recent software source code and hardware design files are available on the project web-pages[14].

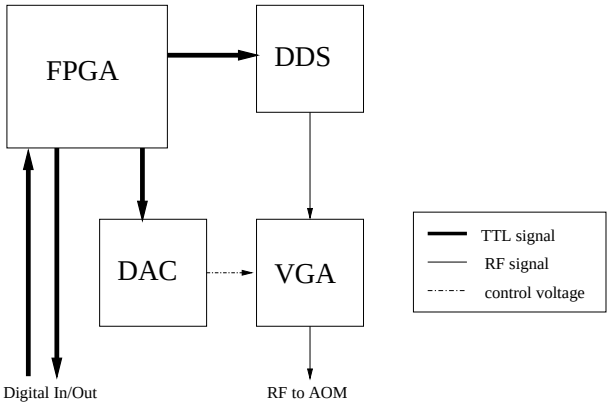

Figure 32: Hardware block diagram of the pulse generator. The core is a programmable logic device (FPGA). Radio frequency pulses are generated by the direct digital synthesizer (DDS). Amplitude modulation of these pulses is realized with the digital to analog converter (DAC) in combination with the variable gain amplifier (VGA). The digital inputs to the FPGA are used as triggers to synchronize the sequence to the mains line phase. The free outputs of the FPGA are used as synchronous digital outputs.

3.4.2 Frequency Generation

To reduce development time and cost, evaluation boards for the direct digital synthesizer, the digital to analog converter, and variable gain amplifier supplied from Analog Devices[15] were used. In order to use more than one frequency source simultaneously, additional addressing electronics ("chain boards") are used for the synthesizer and the digital to analog converter. The variable gain amplifier is controlled by an analog voltage and therefore requires no additional addressing logic. It is possible to address up to 16 different radio frequency channels with four address bits for the synthesizer and the digital to analog

[12] http://www.sourceforge.net
[13] http://www.opensource.org/licenses/bsd-license.php
[14] http://pulse-sequencer.sf.net
[15] http://www.analog.com

converter. The schematics of these chain boards are available on the project web-page[16] .

Direct digital synthesis The direct digital synthesizer may be divided into digital and analog parts. A block diagram of a direct digital synthesizer is shown in Fig. 33. The digital part consists of a phase accumulator and a sine lookup table. The phase accumulator increases its value every time it receives a positive slope on the clock input. The value by which the phase register is increased determines the frequency of the generated sine and is called the frequency tuning word (FTW). From the actual value in the phase accumulator the amplitude is obtained with a sine lookup table. The width of the phase register is 32 bit, but since a 32 bit lookup table would require over 400 million entries, only the 12 most significant bits are used. This constraint has no effect on the accuracy of the frequency, which is still determined by the phase and therefore the 32 bit register. The sine amplitudes are converted into an analog current with a 10 bit digital to analog converter inside the chip. Therefore there is almost no loss of information due to the reduced width of the sine lookup table. On the analog side there are impedance matching and filter networks.

As for all digitally sampled signals, the Nyquist sampling theorem has to be applied on the output of the synthesizer [26]. This theorem states that every frequency f, generated with a sampling frequency f_s, produces an additional mirror frequency $f_m = f_s - f$. This limits the output frequency f to a maximum frequency $f_{max} \leq f_s/2$. The unwanted mirror frequencies have to be filtered out using analog filters. In practice the maximum frequency is determined by the steepness of this output filters to $f_{max} \approx 0.4 \cdot f_s$. A more detailed introduction to direct digital synthesizer operation is given in references [27, 28].

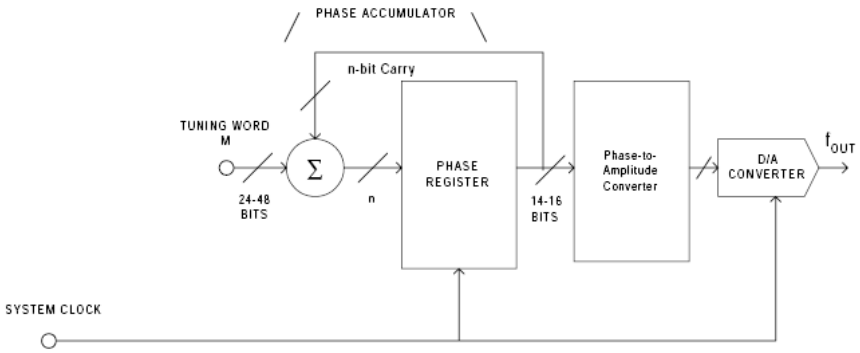

Figure 33: Block diagram of the direct digital synthesizer. It consists of a phase accumulator, a sine lookup table and a digital to analog converter. (image from [27])

Performance of the direct digital synthesizer The main advantage of a direct digital synthesizer over analog frequency generation methods is that the switching between frequencies and even switching on and off the output may be achieved in a few hundred nanoseconds. The frequency drift of the output signal is only dependent on the drift of the reference frequency which may be obtained from a precise frequency standard. Given

[16] http://pulse-sequencer.sf.net

the fact that the phase is controlled digitally and switched with great accuracy and repeatability, phase coherent switching between several frequencies is possible within a few microseconds.

The limitations of direct digital synthesizers are due mainly to the limited resolution of the built in digital to analog converter and appear as spectral impurities and quantized digital to analog converter noise. Another fundamental disadvantage of digital to analog conversion is the presence of mirror frequencies. The higher harmonics generated by non-linearities of the output are also subject to the Nyquist theorem and mirror frequencies of them may be near the fundamental frequency. To estimate the spectrum of the DDS for all possible clock and output frequencies an on-line simulation tool is available at the Analog Devices website[17].

The measured figures of merit for the synthesizer used in the programmable pulse generator are presented in Tab. 5. The frequency resolution of the synthesizer depends on the register width of the phase accumulator and the reference frequency. The minimal frequency step is equivalent to a change in the least significant bit of the frequency tuning word. This leads to the following resolution for a reference frequency of 800MHz:

$$\Delta f = \frac{f_{ref}}{2^{32}} = 0.18\,Hz$$

If a higher frequency resolution is required, it is possible to decrease the reference frequency, however this also decreases the maximum achievable output frequency.

Maximum output frequency	$\approx 350\,MHz$
Minimum output frequency	$< 10\,MHz$
Number of coherent frequencies	16
Frequency switching time	$\leq 150\,ns$
Phase offset accuracy	$2\pi\ 2.4 \cdot 10^{-4}$
Signal to noise ratio	$50\,dBc$
Frequency resolution	$0.18\,Hz$
Maximum output level	$\approx -1\,dBm$

Table 5: Figures of merit for the radio frequency generation of the programmable pulse generator for a clock frequency of 800MHz.

Phase coherent switching For coherent qubit manipulation the phase of the laser light is crucial as this determines the axis of rotation within the XY plane of the Bloch sphere. The phase reference for a particular transition is set by the first laser pulse exciting that transition. All subsequent pulses have to be phase-coherent with respect to the first pulse in order to achieve fully controllable coherent state manipulation. As the laser is driven by an AOM, the phase may be controlled by setting the phase of the radio frequency driving the modulator. Additionally a single sequence may include pulses on more than one transition (for example carrier and sideband transitions) therefore phase coherent switching of multiple frequencies is required within one sequence. A phase offset is required to realize

[17] http://designtools.analog.com/dtDDSWeb/dtDDSMain.aspx

rotations around the Z axis of the Bloch sphere. The synthesizer has only one phase accumulator built in, therefore phase continuous switching between different frequencies is not possible with a single synthesizer. Fig. 34 gives an overview of the different switching modes.

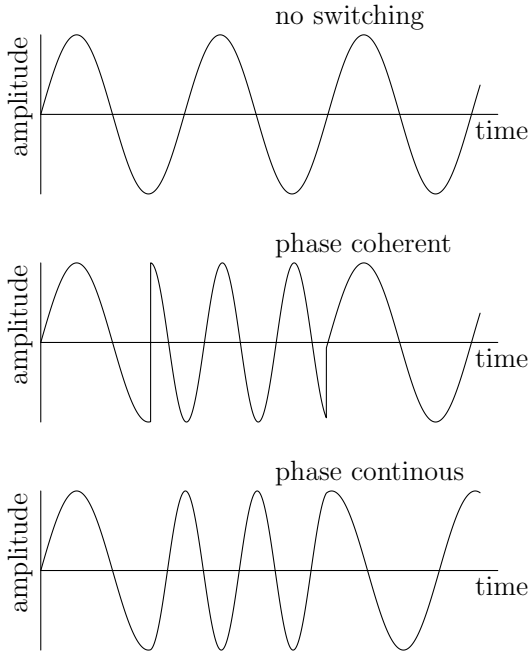

Figure 34: Different switching methods. For phase coherent switching the phase is set to the value as if there was no switching at all. For phase continuous switching there are no phase jumps.

Because the synthesizer has only a single phase accumulator, keeping track of the phase of multiple frequencies is handled by the FPGA which has 16 phase accumulators similar to the ones built into the synthesizer. These phase accumulators have the same width as the phase accumulator in the synthesizer. The FPGA can only be clocked with maximum frequencies of around 100 MHz and therefore the FPGAs frequency tuning word is always the tuning word of the synthesizer multiplied by eight. In a phase coherent frequency switching event, the content of one of these registers is written into the phase accumulator of the synthesizer. As there are different relative phases required for a single frequency it is possible to add a constant value to the current phase accumulator in the FPGA. The timing of the writing process determines the quality of the phase coherent switching. As long as the delay between reading out the phase accumulator in the FPGA and setting the phase accumulator in the synthesizer is constant for every switching process, the delay may be left uncompensated as it leads to a constant phase offset. For writing the phase word to the synthesizer the 32 bit word is split into four eight bit words. To keep the phase word constant during the switching event, the value of the phase accumulator in the FPGA is copied into a dedicated register. This register is then transferred to the synthesizer. After this writing process the synthesizer takes over the new phase word with the rising slope of a digital control signal generated by the FPGA. This process ensures a constant time

delay between writing the phase accumulator into the register in the FPGA and updating the phase register in the synthesizer.

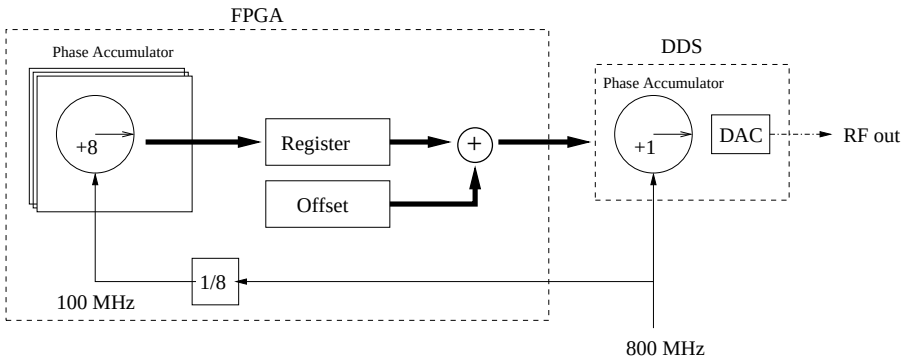

Figure 35: Block diagram of the phase registers inside the FPGA. The FPGA clock frequency is the synthesizer reference clock frequency divided by eight. A phase offset is added to the value of the phase register and the sum is written to the synthesizer.

Amplitude shaping In order to create arbitrary pulse shapes for reducing the off-resonant excitation, a variable gain amplifier controlled by a digital to analog converter is used. The digital to analog converter is itself controlled by the FPGA. An amplitude shaped pulse is generated in five steps:

- Phase coherent switching to the desired frequency and relative phase.
- Generation of the rising slope with the digital to analog converter.
- Wait cycles for realizing the desired pulse length.
- Generation of the falling slope with the digital to analog converter.
- Switching off the synthesizer.

The variable gain amplifier is an analog devices AD8367 which is logarithmic in control voltage. Therefore the digital to analog converter has to compensate this behaviour. This is done while compiling the sequence in the experiment control computer. The digital to analog converter is an AD9744 which has a resolution width of 14 bits and a maximum clock rate of 100 MHz. Addressing of multiple converters is realized with chain boards which enable the clock of the converter depending on the addressing word. The resolution of the amplitude shaping is $0.01\,dB$ (the resolution is given in dB because the VGA is logarithmic in control voltage). Therefore the resolution of the (linear) pulse shape changes with the actual radio frequency power used.

3.4.3 The FPGA Core

The design of the FPGA system of the programmable pulse generator may be subdivided in three different layers: the hardware, the firmware and the software. The hardware layer

was designed by Paul Pham for his Master's thesis at MIT [25]. The firmware layer controls the behaviour of the FPGA. The firmware currently used was also written by Paul Pham, whereas future versions are likely to include new components written by the author. The software layer is running on the experiment control computer and handles the generation and transmission of the machine code. It was written in a collaboration between Paul Pham and the author.

The hardware is designed to be able to generate accurate digital pulses. The Low Voltage Differential Signaling (LVDS) logic standard is used for external digital signals. This standard was defined by National Semiconductor[18] and is widely used for high speed long distance digital signal transmission. Due to its differential nature it is very insensitive to electrical noise coupled into and reflections generated inside the transmission lines. The board has different options for reference clock signals. The two independent clock inputs of the FPGA may be either connected to external radio frequency connectors, connected to a clock derived from the external bus or connected to one of the two quartz oscillators placed on the board. To increase the possible program size an external memory chip is available on the board.

The firmware may be divided in the following blocks, as shown in Fig. 36

- PTP pulse transfer protocol: For transferring data between the computer and the FPGA.
- PCP pulse control processor: The core with the timing and program control logic.
- Interfaces to the external and internal peripherals.

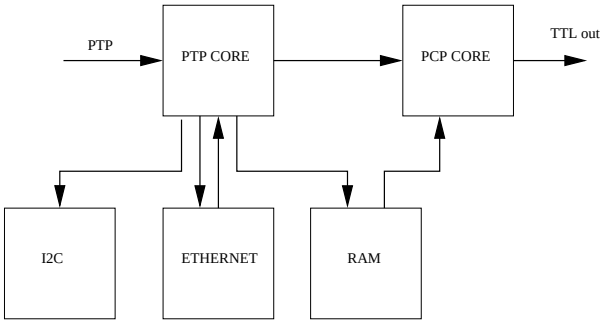

Figure 36: Simplified block diagram of the FPGA firmware. The transfer core handles the communication with the experimental control computer and writes the byte code to the memory(RAM). It also handles the start and stop commands for the timing core (PCP). The timing core reads out the program from the memory and decodes and executes it. The I2C core is controlled directly by the transfer core.

The pulse transfer protocol is a custom protocol for controlling the programmable pulse generator. It is based on the User Datagram Protocol (UDP) which is a standard Internet protocol. The part of the firmware which implements this protocol is denoted in the following as the transfer core. The protocol supports directly the connection and

[18] http://www.national.com

synchronization of several FPGA boards. The most important commands are shown in Tab. 6.

Name	Description
Discover	Searches for connected programmable pulse generators
I2C	Sends data to the general purpose I2C Bus
Start	Starts the current sequence.
Stop	Stops the timing core to initiate a new data transfer.
Write	Writes data to the memory of the programmable pulse generator.

Table 6: Important commands of the pulse transfer protocol

The full specifications of the pulse transfer protocol may be found in chapter 4 of Paul Pham's Master thesis [25] or in appendix(C). The transfer core extracts the machine code from the received data and writes it to the memory (RAM) of the programmable pulse generator. After receiving the data the sequence is initiated with the start command.

After receiving the start command, the pulse control processor (PCP) fetches the first instruction from the memory. This part of the firmware is denoted as the timing core. It decodes and executes the single commands of a program. In addition to basic program control structures, the instruction set also contains commands for setting the digital output and for phase coherent switching. Although the instruction set shows parallels with a microprocessor there are no general purpose registers, arithmetic or logic functions implemented. This leads to the fact that the only way to influence the running program is through trigger inputs. Small changes require a full recompilation. The most important commands are shown in Tab. 7. The phase accumulators required for phase coherent switching as described in section(3.4.2) are also part of the timing core.

The outputs on the external bus are separated in digital TTL outputs of the programmable pulse generator and control signals for the synthesizer and the digital to analog converter. In total the external bus is 64 Bits wide, where 20 bits are available for digital TTL outputs. For the implementation of the wait command, the timing core uses a register which is loaded with the number of clock cycles which correspond to the desired waiting time. For each rising edge of the clock the value of this register is decreased by one. The next command is not executed until the register value is zero.

Additional information about the pulse transfer protocol and the programming of the programmable pulse generator is given in appendix(C).

Name	Description
Pulse	Generates a digital pulse on the external bus.
PulsePhase	Writes the content of a phase accumulator to the external bus.
Jump	Jumps to a different memory address.
Branch	Jumps to a different memory address depending on triggers.
Wait	Waits for a certain number of clock cycles.

Table 7: Important machine code commands of the programmable pulse generator

3.5 The Software

3.5.1 Overview

This section gives an overview over the software which is used for experiment control of the $^{40}Ca^{+}$ experiment. The main experiment control program is written in the graphical programming language of LabView. It was written by Wolfgang Hänsel and a detailed description would be beyond the scope of this thesis. This section will concentrate on the part of the experimental control software which generates and transfers the machine code for the programmable pulse generator. This software was written entirely in the Python open source programming language[19]. It will be denoted as the Python server. An overview of the total experiment control software is given in Fig. 37. The Python server source code is available on the pulse sequencer project homepage[20]. Communication between the LabView program and the Python server is realized with a standard network protocol (TCP[21]) to ensure maximum portability. In the current setup the Python server and the LabView program run on the same computer and communicate over the Ethernet loopback interface. The Python server is itself divided into three different layers. It receives a human readable pulse sequence, converts this into machine code for the timing core inside the FPGA and then transmits this to the programmable pulse generator.

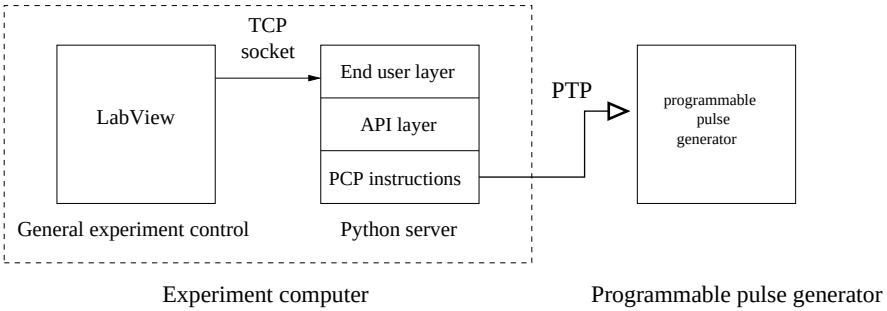

Figure 37: Experiment control software. The LabView control software and the Python server are connected via a TCP connection. The Python server generates byte code for the FPGA and transmits it via the pulse transfer protocol.

3.5.2 The Python server

The different Layers of the Python server are shown in Tab. 8. A layer provides functions to the layer above and uses functions from the layer below. The end user layer receives functions directly from the LabView program and converts this to API layer commands. Examples of functions at the end user layer layer would be "single qubit rotation" or "side-band cooling". The application programming interface (API) layer converts this functions to commands for the programmable pulse generator. Examples of functions at the API layer are "switch on synthesizer" or "set digital output to level high". The pulse transfer

[19] www.python.org
[20] http://pulse-sequencer.sf.net
[21] Transmission Control Protocol

protocol is implemented in this layer. It is also used for writing small programs to debug the hardware. The compiler layer generates the binary machine code from these API commands.

Name	Description
End user	Receives human readable pulse sequence.
API	Converts pulse sequence to events for the programmable pulse generator
compiler	Machine code generation for the timing core

Table 8: The Python server software layers.

There are two different ways of describing a pulse sequence in the end user layer. They are called the "parallel mode" and the "sequential mode". The parallel mode is used in the $^{40}Ca^{+}$ experiment whereas the sequential mode is used by all other experiments. In the parallel mode the transmitted data is directly executed by the Python server whereas in sequential mode the LabView program sends a command string with the file name and the parameters of the current sequence. The difference between the two modes is described in appendix(B).

The syntax of the following program examples is not strictly correct but instead shows the concept of the different layers. In Algorithm(1) a simple end user layer program is shown. It consists of a simple experimental sequence with Doppler cooling, a single qubit rotation and qubit detection. The corresponding API layer program is shown in Algorithm(2). While the end user layer program is self explanatory the API layer program needs some further explanations.

The begin_sequence() function is mandatory at the beginning of all sequences. It generates the internal variables used for the API and the compiler layer and additionally generates events for initializing the external hardware. The begin_finite_loop() marks the start of the sequence to be repeated. The phase accumulators in the FPGA are initialized with the initialize_frequency(f_carrier) command. The set_ttl() and switch_frequency() commands are used to generate the digital and radio frequency pulses. The end_finite_loop(100) command marks the end of the repeated sequence. The end_sequence() command starts the compilation and the transmission of the sequence. A complete list of all available commands is given in the appendix of this thesis.

Algorithm 1 Example end-user program

```
doppler_cooling()
R729(1,1,0,Carrier)
detection()
```

3.5.3 Limitations

One major limitation of the Python server is the compilation speed. Amplitude shaped pulses need 100 or more digital to analog converter events. When the amplitude shape is calculated for every pulse used in the sequence this leads quickly to a large sequence which

Algorithm 2 The API program generated from the sequence shown in Algorithm(1)

```
begin_sequence()
begin_finite(loop)
initialize_frequency(f_carrier)

# doppler cooling
set_ttl(["866 sw","397 dopp","397 sw",1)
wait(1000)
set_ttl(["866 sw","397 dopp","397 sw",0)

# single qubit rotation
switch_frequency(f_carrier,0)
rising_slope(shape_time,amplitude)
wait(t_pulse)
falling_slope(shape_time,amplitude)
switch_off(f_carrier)

# detection
set_ttl(["866 sw","397 det","397 sw",1)
wait(2000)
set_ttl(["866 sw","397 det","397 sw",0)

end_finite_loop(count=100)
end_sequence()
```

takes over one minute to compile. In order to work comfortably the compile time should not be much longer than one second. Therefore the amplitude shapes are compiled once and stay in the FPGA memory until a new shape is compiled. In the program they are then invoked as subroutines. They are not overwritten when a new sequence is generated, and this leads to short compilation times when the parameters of the amplitude shape is not altered. If the parameters of the amplitude shape has to be altered this leads to long compilation times and makes system calibration tedious.

Another bug of the programmable pulse generator is that when a new sequence is loaded, sometimes the first command of the sequence may not be executed correctly. This is due to a known bug in the FPGA firmware. A workaround is simply sequence recompilation.

3.6 Future development of the programmable pulse generator

The programmable pulse generator is currently in further development to enable a more general use. The instruction set is being completely rewritten to include more specialized commands for digital to analog converters and direct digital synthesizer control. The lack of a counter, for example for counting fluorescence photons, is a major missing feature which will be added in the near future. Experiments with pulses depending on a measurement during the sequence will become possible without the use of external hardware. In general the development will lead to a more flexible core, where arithmetic and logic operations

can be used to create the amplitude pulse shapes at runtime thus saving compilation time.

There is also an effort to develop custom boards for radio frequency generation. In addition, by combining the data bus for the digital to analog converter and the direct digital synthesizer, additional digital output channels may be obtained. The goal is to increase the count of digital outputs from 21 to 32.

4 Experimental results

4.1 Functionality tests of the programmable pulse generator

Prior to integrating the programmable pulse generator into the experimental setup, its features had to be tested thoroughly. The emphasis of these tests was on characterizing the radio frequency outputs and the phase coherent switching. Testing the digital outputs and the communication with the experiment computer was mainly done by Paul Pham and is not described here.

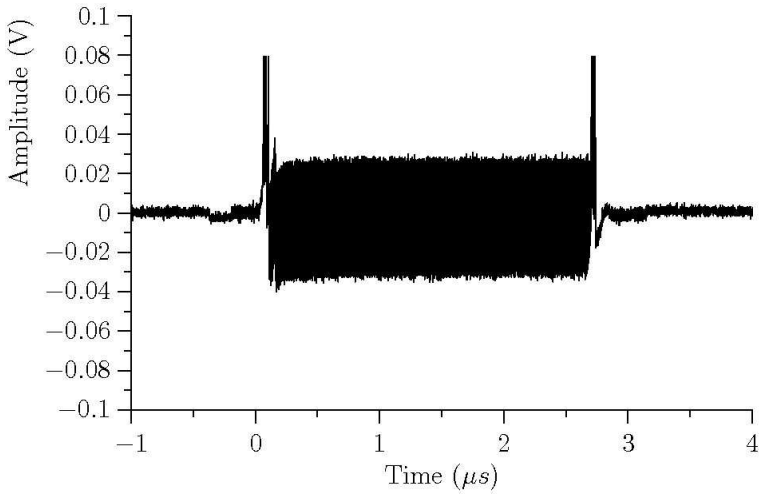

Figure 38: Radio frequency pulse generated by the direct digital synthesizer. This pulse was measured with an oscilloscope directly at the output of the synthesizer.

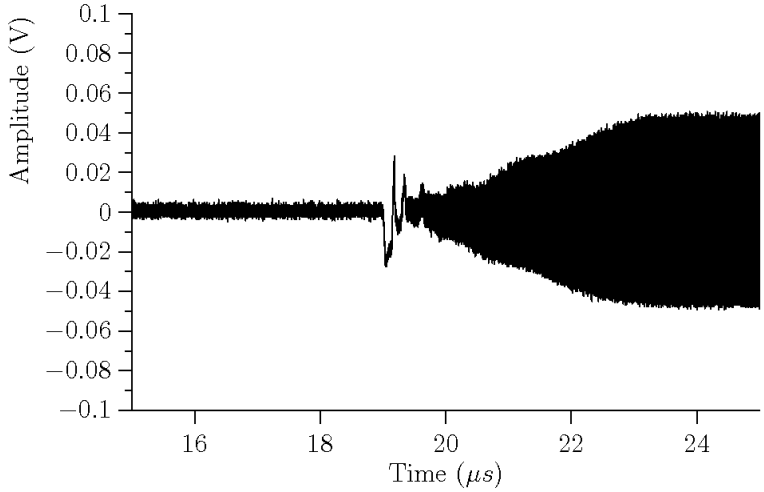

Figure 39: Radio frequency pulse with a Blackman shape. The spikes at the beginning of the pulse are suppressed by switching on the variable gain amplifier delayed.

First the quality and the switching speed of the radio frequency signal from the direct digital synthesizer was analyzed. In Fig. 38 a rectangular radio frequency pulse is shown. The noticeable spikes at the beginning and the end of the pulse are caused by the analog output filter on the direct digital synthesizer evaluation board. The spikes resemble

the step response of this output filter. The effect of theses spikes may be minimized by switching the direct digital synthesizer on while the variable gain amplifier is still switched off. The variable gain amplifier is switched on after the spike at the beginning of the pulse vanishes. Analogously the variable gain amplifier is switched off before the synthesizer. The beginning of a Blackman shaped pulse with a suppressed spike is shown in Fig. 39. The spikes are attenuated by the maximum achievable attenuation with the variable gain amplifier which is 40dB. The spike in Fig. 38 is clipped by the digital oscilloscope and therefore the attenuation seems to be only a factor of 4 whereas it is attenuated by a bigger amount. Furthermore the Fourier transform of these spikes contains only frequencies lower than 50MHz and therefore does not affect the modulated laser light as the modulator has a center frequency of 270MHz and a bandwidth of 50MHz.

4.1.1 Spectrum of the direct digital synthesizer output

Generally the spectral quality of a radio frequency source has to be measured when the source operates in continuous wave mode. The spectrum of a radio frequency pulse as shown in Fig. 38 is broadened due to its finite length and therefore it is not possible to measure the spectral quality of the source with a short pulse. As mentioned in section(3.4.2) the spectral quality of the direct digital synthesizer is direct related to the quality of its clock signal. The clock is generated by a high quality Marconi synthesizer[22] which is synchronized to an oven controlled crystal oscillator[23]. The Marconi synthesizer is specified with a phase noise of better than 116dBc/Hz at an offset of 20kHz. According to its datasheet the phase noise of the direct digital synthesizer can be as good as 150dBc/Hz for an offset of 100kHz. In our setup the phase noise of the radio frequency output is then determined by the phase noise of the reference clock which is generated by the Marconi synthesizer. The phase noise of the synthesizer output was not measured as there was no suitable measurement device available.

The spectrum of the direct digital synthesizer output is shown in Fig. 40 and Fig. 41. A measure of the spectral quality is the spurious free dynamic range (SFDR) which is the ratio of the power of the carrier signal to the power of the largest spurious signal. From the narrow spectrum the narrow band spurious free dynamic range is measured to be 70dBc whereas the wide band spurious free dynamic range is 55dBc. The values given in the datasheet are 72dBc for the narrow band and 52dBc for the wide band spurious free dynamic range. The highest side-lobe in the wide band spectrum is the mirror frequency of the second harmonic of the synthesizer's digital to analog converter output. The origin of the largest spurious signal may vary for different output and clock frequencies. The carrier frequency for this measurement was 268MHz and the second harmonic frequency is 536MHz. With a clock frequency of 800MHz the mirror frequency is at $f_{DAC,2} = f_{clock} - 2 \cdot f_{carrier} = 264\,\mathrm{MHz}$.

[22] Marconi 2032A
[23] Oscilloquartz OCXO8600

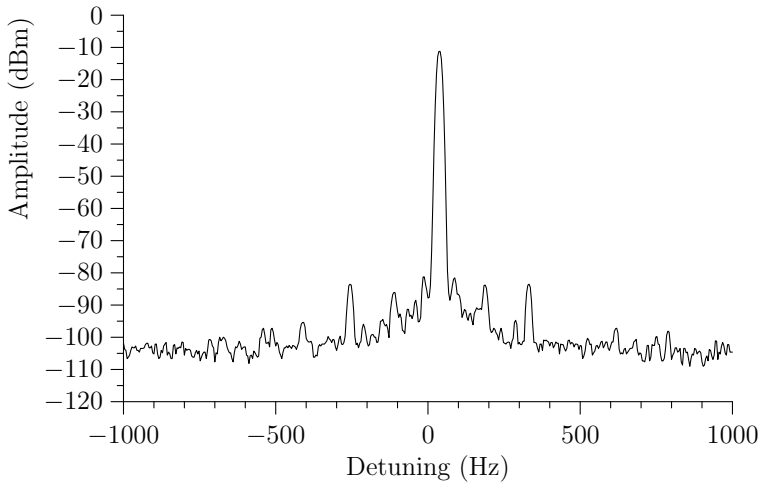

Figure 40: Narrow spectrum of the DDS output. The largest spurious signal has a power of -80dBm whereas the carrier signal is at -10dBm.

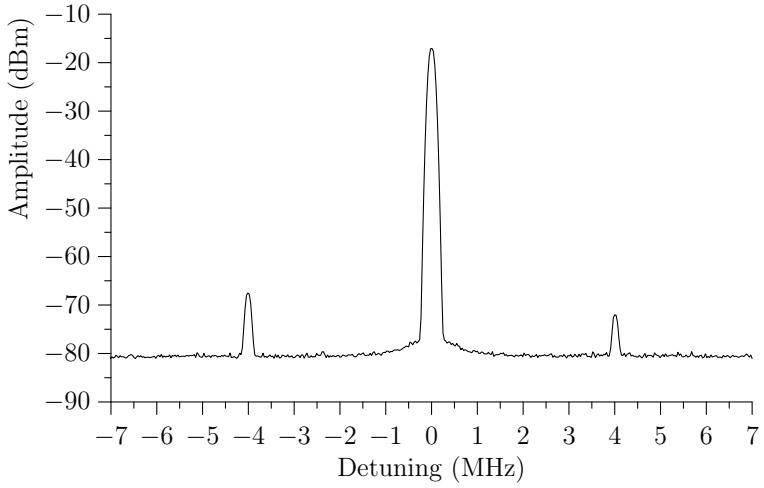

Figure 41: Wide spectrum of the DDS output. The largest spurious signal is the mirror frequency of the second harmonic of the carrier frequency.

4.1.2 Characterization of the phase coherent switching

For the characterization of the quality of phase coherent switching, the phase difference between two radio frequency sources is observed with a phase detector. For the measurement two independent direct digital synthesizer outputs from one programmable pulse generator were connected to a mixer[24]. A mixer multiplies the two input signals $u_1(t) = U_1 \cos(\omega_1 t + \phi)$ and $u_2(t) = U_2 \cos(\omega_2 t)$ and therefore produces a component with the frequency difference $U_{out} = U_1 \cdot U_2 \cdot \sin((\omega_1 - \omega_2) \cdot t - \phi)$ and a component with the sum of the two frequencies. In this measurement the component oscillating with the frequency $\omega_1 + \omega_2$ is filtered out by a low pass filter and may therefore be neglected. In the measurement setup one synthesizer was programmed to generate a continuous wave radio frequency output at a constant frequency f_0. The other synthesizer was switched on and

[24] Mini Circuits ZAD-1

off at the same frequency using phase coherent switching. The sequence used for testing the phase coherent switching consists of three pulses where the phase of the first pulse is slightly different to the phase of the other two pulses. By measuring the difference voltage between the second and the third pulse, the quality of the phase coherent switching process can be determined. Fig. 43 shows the output of the mixer for a phase difference between the first and the second pulses of $\phi_{1,2} = \pi/200$. This corresponds to a voltage difference of approximately 1.3mV. The phase difference between the second and the third pulse is set to zero. The voltage difference between Pulse2 and Pulse3 can be estimated to be below 0.3mV and therefore the error of the phase coherent switching may be expressed as

$$\Delta\phi \leq \frac{0.3mV}{1.3mV}\,\frac{\pi}{200} = \frac{\pi}{800} \quad .$$

The fast glitches shown in Fig. 43 are ring down effects in the low pass filter and are caused by neither the direct digital synthesizer nor the variable gain amplifier. If these ring down effects would be caused by the synthesizer they should also be visible in Fig. 38 or Fig. 39 which is not the case.

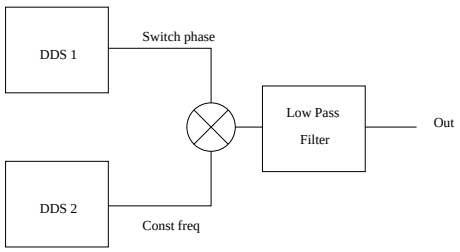

Figure 42: Electronic setup for measuring phase coherent switching

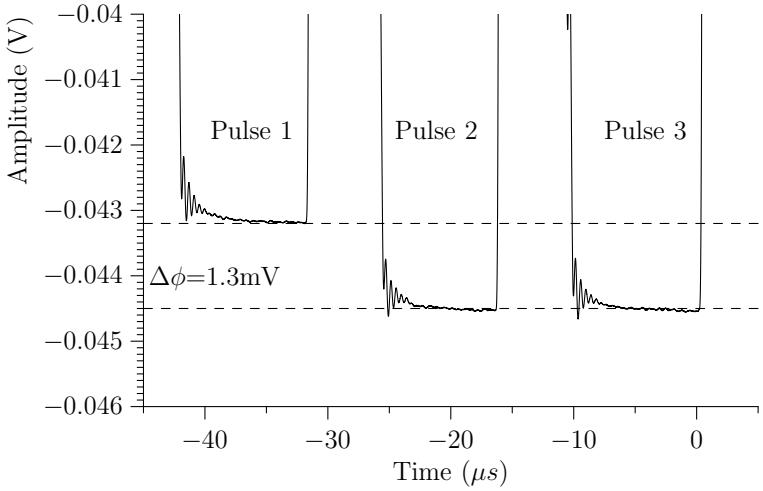

Figure 43: Phase coherent switching events. The phase difference between Pulse1 and the succeeding pulses is $\Delta\phi = \pi/200$.

4.1.3 Calibration of the radio frequency power

The radio frequency output power of the programmable pulse generator is determined by the digital to analog converter output voltage which controls the variable gain amplifier. The amplification of the variable gain amplifier is logarithmic in control voltage. The dependence of the radio frequency output power on the digital to analog converter value is shown in Fig. 44. The transferred data from the LabView experiment control program contains the radio frequency amplitudes in linear scale between 0 and 1. For compatibility with the previous radio frequency setup, where the signals were generated with Marconi synthesizers, a transferred amplitude of 1 corresponds to the former maximum achievable output power (13dBm). This power is set with external fixed attenuators. The hardware configuration of the radio frequency channels is shown in Fig. 46. The acousto optical modulator is driven with a maximum of 20dBm to prevent thermal effects due to power dissipation. According to the datasheet the variable gain amplifier has a dependency on the control voltage of $10dB\,/200mV$. The control voltage is calculated from the digital to analog converter value y as $U_{control} = U_{max} \cdot y/(2^{14} - 1)$. As the variable gain amplifier has a poor switching performance if the control voltage is at $0V$ previous to a pulse, a minimum voltage is applied. The function used for calibrating the variable gain amplifier is

$$y = a + b \cdot \log_{10}(x + 10^{\frac{c-a}{b}}) \quad .$$

When inserting the values found in the datasheet this leads to following coefficients.

$$\begin{aligned} a &= 14300 \\ b &= 3200 \\ c &= 1500 \quad . \end{aligned}$$

The remaining non linear behaviour shown in Fig. 45 is due to saturation effects in the acousto optical modulator.

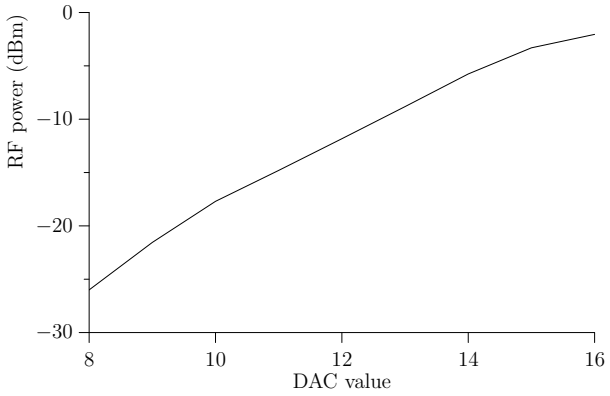

Figure 44: RF output power in dBm as a function of the DAC value.

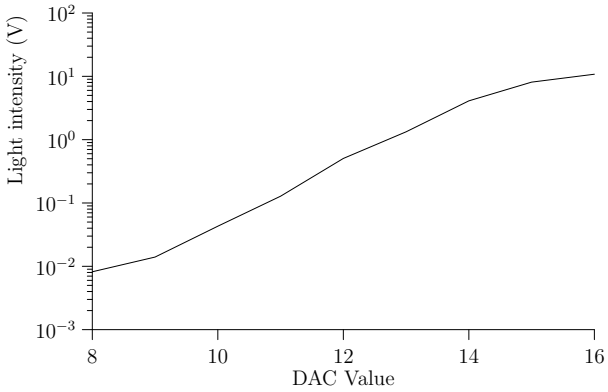

Figure 45: Light intensity measured on a photo diode as a function of the DAC value.

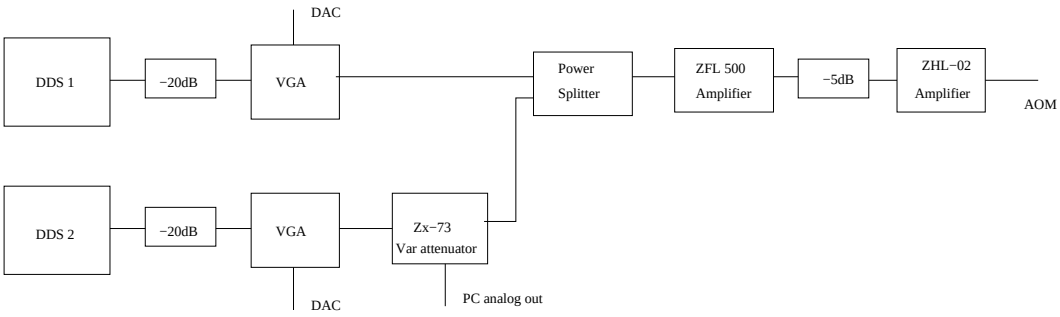

Figure 46: 729nm AOM radio frequency setup.

4.2 Starting up and calibrating the experiment

The first step when operating the experiment is loading the ions into the trap. A beam of neutral ^{40}Ca atoms is generated by an oven inside the vacuum vessel. The atoms are then ionized with a two step photo-ionization process inside the trapping region [16]. For high loading rates the photo-ionization beams are spatially superimposed with the Doppler cooling beam and therefore the ions are cooled immediately after ionization. After loading the ions, the lasers are adjusted for best spatial overlap with the ion chain. This is achieved by weakly driving the $S_{1/2} \rightarrow P_{1/2}$ transition and maximizing the fluorescence. To achieve optimal conditions for Doppler cooling the laser power is adjusted to the saturation power of the cooling transition and detuned about half the transition linewidth. The frequency of the 866nm repump laser is adjusted by maximizing the count rate while switching on the detection 397nm laser. For adjusting the frequency of the 854nm laser the 729nm laser is shone in. Without 854nm light this decreases the count rate by a factor of two. The frequency of the 854nm laser is then set to the point where the fluorescence is maximized. These adjustments are made in a continuous fluorescence monitoring mode, whereas for coherent manipulation of the qubit the lasers are switched on and off during an experimental cycle. The procedure for starting up the experiment is discussed in more detail in the PhD theses of Hanns Christoph Nägerl [13] , Harald Rohde [29], Christian Roos [30] and Mark Riebe [16].

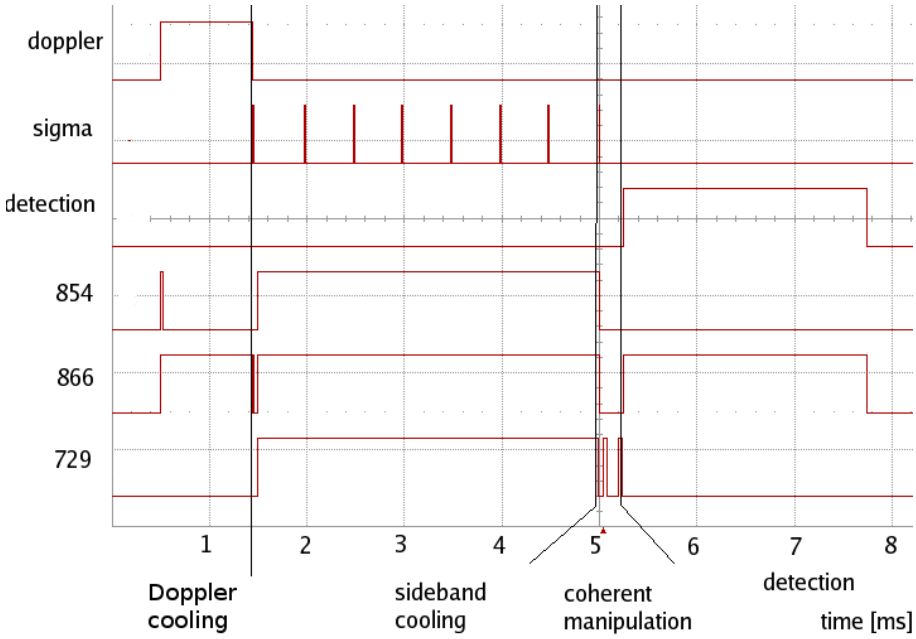

Figure 47: Scheme of a typical pulse sequence. The main parts are: state preparation, Doppler cooling, sideband cooling, coherent manipulation and state detection.

The following adjustments are made using a pulsed scheme, where a typical experimental sequence is shown in Fig. 47. The first 854nm light pulse transfers any population left in the $D_{3/2}$ state from the previous sequence to the $P_{3/2}$ state. From this state the ion decays quickly into the $S_{1/2}$ state. Doppler cooling is carried out by switching on the

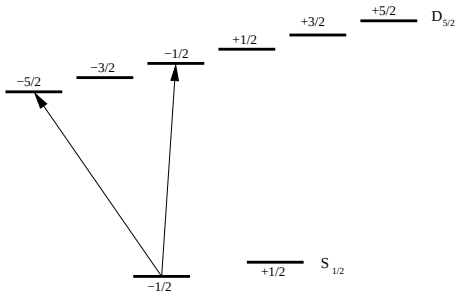

Figure 48: Zeeman sublevels of the $S_{1/2} \rightarrow D_{5/2}$ transition. The arrows indicate the two main carrier transitions.

397nm and 866nm lasers simultaneously. For optical pumping to the $m = -\frac{1}{2}$ Zeeman level, several short pulses of σ^- polarized 397nm light are applied. Sideband cooling is done with a 729nm laser beam which is red detuned by the axial trap frequency. Additionally the 854nm repump laser is switched on simultaneously to increase the cooling rate and short σ^- pulses of 397nm light are applied to ensure that only the $m = -\frac{1}{2}$ Zeeman level is populated. Coherent manipulation of the qubit consists of a series of 729nm laser pulses. For state detection 397nm and 866nm light are switched on simultaneously and the ion's fluorescence is detected with the photo multiplier and the CCD camera.

The focus of the 729nm laser is optimized with respect to the ions positions with a lens mounted on a digitally controlled translation stage and the electro optical deflector. After adjusting the spatial positions of the laser beams, the exact frequencies of the carrier transitions are determined. The Zeeman sublevels used in our experiments are shown in Fig. 48. For determining the magnetic field strength and the unshifted transition frequency, the two transitions shown in Fig. 48 are used. Due to temperature changes of the reference cavity for the 729nm laser and slow fluctuations in the magnetic field, the laser frequency has to be adjusted at the start of every experimental run. Once the system is completely set up, the evolution of the magnetic field and the cavity frequency is traced by periodic measurements at a typical interval of 30 - 200s. The typical value for the cavity drift is around $1\,Hz/s$ and for the magnetic field drift $1 \cdot 10^{-7} G/s$.

Then the ion positions and Rabi frequencies are measured. For determining the Rabi frequency, a sequence with a resonant 729nm pulse and varying pulse length is used. The Rabi frequency is obtained by fitting a sinusoidal curve to the resulting data. The Rabi frequency is proportional to the amplitude of the electric field and therefore the relationship between the Rabi frequency and the optical power is $\Omega \sim \sqrt{P_{optical}}$. Here one has to bear in mind that the AOM of the 729 laser is used in double pass configuration and so the optical power scales with the square of the radio frequency power and this finally yields $\Omega \sim P_{RF}$. This means that the Rabi frequency should double approximately with every 3dB decrease in radio frequency power. As can be seen in Tab. 9 this is not exactly the case. The reason for this are non linear effects in the AOM. These non linearities are also observed when calibrating the radio frequency power in section(4.1.3).

Power	Rabi 2π Time
2dB	2.9 μs
5dB	1.62 μs
8dB	1.02 μs

Table 9: Rabi frequencies for different RF pulse powers. The dB values correspond to the Marconi setup and are therefore relative units.

4.3 Characterizing off-resonant excitations

In this chapter a method for measuring off-resonant excitations is presented. In a sequence for quantifying these excitations the coherent manipulation consists of a single 729nm pulse, with variable duration, detuning, RF power and pulse shape. For a single data point the length of this pulse is varied. A sinusoidal curve with varying offset, frequency and amplitude is fit to this data as shown in Fig. 49. In most cases only the amplitude of the sine is used for further processing. For verifying the simple results obtained for rectangular pulses in section(2.1) the laser detuning with respect to the qubit transition was varied. According to the theoretical results the population oscillation dependence on the detuning should be $p = \frac{1}{1+(\frac{\Delta}{\Omega})^2}$, which is verified by the fit shown in Fig. 50. From this fit the Rabi frequency is determined to be $\Omega = 2\pi\,182kHz$. The increasing tail of the measured data which starts at a detuning of about 900kHz corresponds to the first motional sideband. As the simulations are based on a a two level system the sidebands are not taken into account.

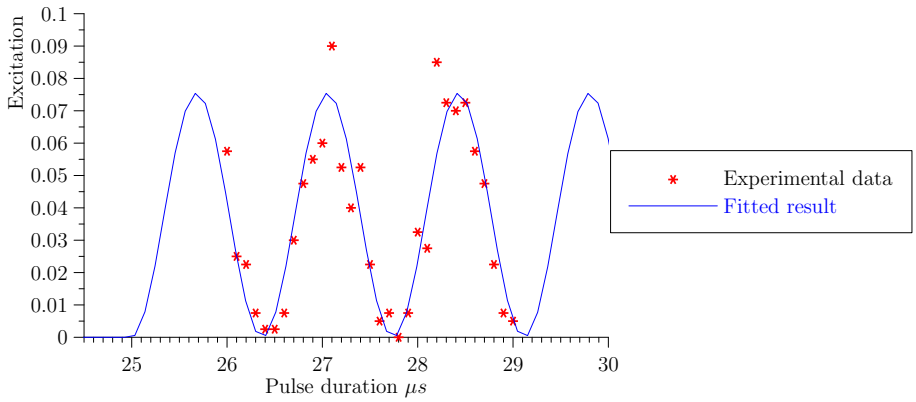

Figure 49: Off resonant excitations with a fitted sine function. The parameters are $\Omega = 2\pi\,182\,\text{kHz}$ and $\delta = 700\,\text{kHz}$.

4.3.1 The effect of pulse shaping

In the following the amount of off-resonant excitations for rectangular pulse and amplitude shaped pulses are compared. The Blackman shape defined in section(2.2) was used for the following experiments. In Fig. 51 the off-resonant excitations as a function of the detuning are compared for two different ramp durations. As expected the off-resonant excitations vanish faster with increasing ramp durations.

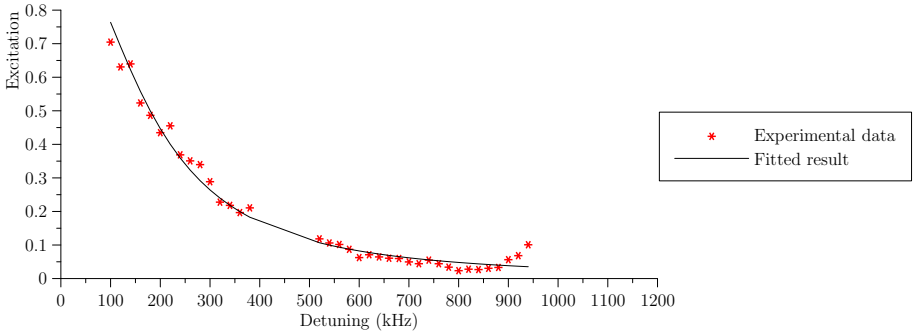

Figure 50: Off-resonant excitations as a function of the detuning for a rectangular pulse. The fit yields a Rabi frequency of $\Omega = 2\pi\, 182kHz$.

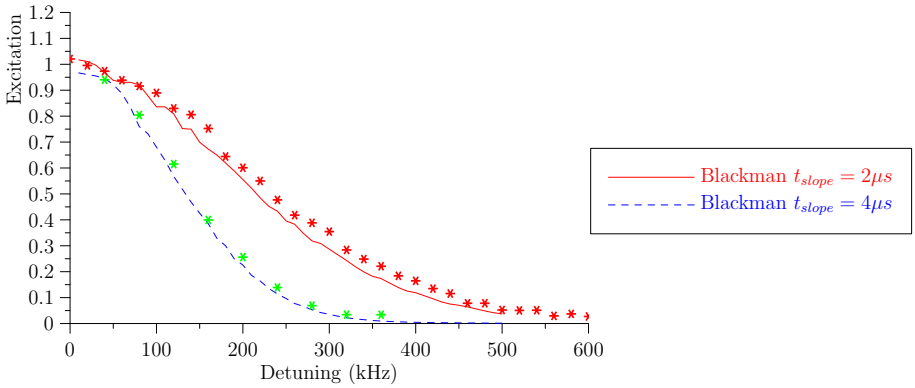

Figure 51: Off-resonant excitations as a function of the detuning for different ramp durations. The solid lines are simulations with a Rabi frequency of $\Omega = 2\pi\, 340\,\mathrm{kHz}$.

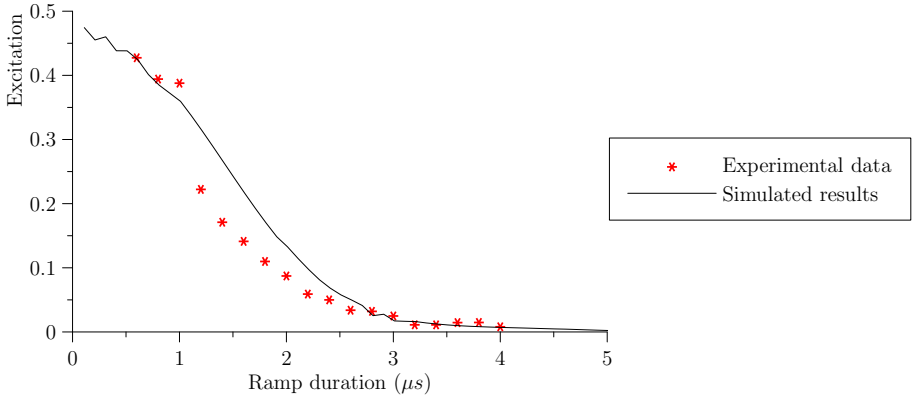

Figure 52: Off resonant excitations as a function of the ramp duration with constant detuning. The only parameter used for the simulation was the Rabi frequency $\Omega = 2\pi\, 380kHz$

In Fig. 52 the lengths of the rising and falling slope are varied for a constant detuning of $\delta = 2\pi\,400\,\mathrm{kHz}$ and a Rabi frequency of $\Omega = 2\pi\,140\,\mathrm{kHz}$. Theory and experiment are in good agreement. Therefore, it is possible to use the simulated values for the typical parameters used in the experiment. Simulations predict off-resonant excitations of $P_D = 3.5\cdot10^{-5}$ when driving a sideband transition with Blackman shaped pulse with $t_{slope} = 3\mu$s and an axial trap frequency of around 1MHz and a Rabi frequency of 200kHz . Due to projection noise and imperfect preparation of the initial state, this low excitation cannot be measured.

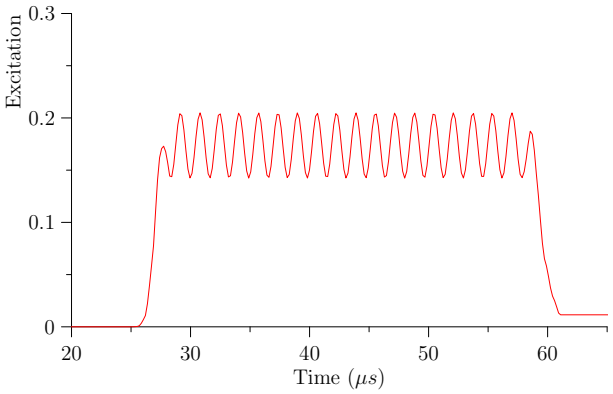

Figure 53: Simulated time evolution of a shaped pulse.

It was shown above that the final state of the time evolution can be predicted with numerical simulations. Further comparison of the experiment to the simulations is accomplished by investigating the entire time evolution. However, it is difficult to access the time evolution during the rising and falling slopes experimentally. Nevertheless it is easy to generate a pulse which consists of only the rising slope and the constant plateau. With this method the time evolution between the rising and the falling slope is accessible. As can be seen in Fig. 53 this evolution is a sinusoidal with constant offset. Therefore amplitude and offset of a sine function are fit to the measured data and the corresponding numerical simulations. In Fig. 54 the doubled amplitude and the offset of the fitted sine are shown for various Rabi frequencies and a ramp duration of $6\,\mu$s. It can be seem that for a Rabi frequency of 160kHz the offset converges to a constant value for large detunings. As stated in section(2.1.5) the state of the qubit is always the corresponding dressed state. In this measurement this would mean, that the amplitude goes to zero while the offset converges to a constant value which corresponds to the dressed state in the qubit basis. For a Rabi frequency of 160kHz the simulations and the experimental results are in good agreement. For a Rabi frequency of 623kHz the amplitudes are not predicted correctly. This is due to decoherence which arises from laser intensity fluctuations, which increases with the laser power due to thermal effects of the acousto optical modulator. Describing these effects in detail is beyond the scope of this thesis.

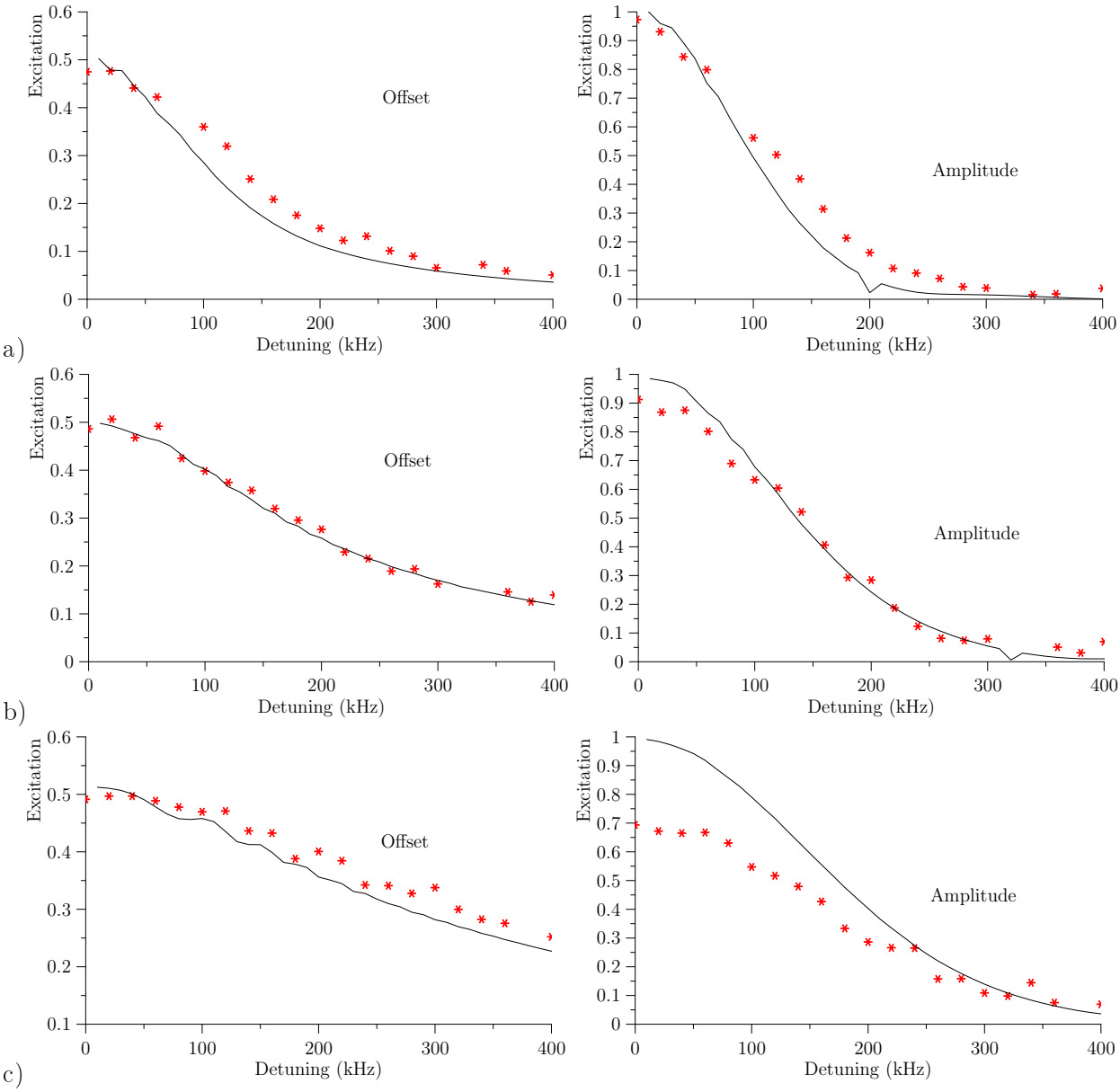

Figure 54: Simulated and measured offset and amplitude without the falling slope for different Rabi frequencies: a) $\Omega = 2\pi\,160\,\text{kHz}$, b) $\Omega = 2\pi\,340\,\text{kHz}$, c) $\Omega = 2\pi\,613\,\text{kHz}$

4.3.2 The influence of the initial state

In the experiment, different initial states are created by a resonant pulse with a given length, and afterwards an off-resonant pulse was applied. The excitation probability for the D state after a pulse with the duration $t = T_{2\pi}\frac{\theta}{2\pi}$ is $P_D = \sin^2(\frac{\theta}{2})$. For evaluating the influence of the off-resonant excitation the duration of the off-resonant pulse was again varied. Only the amplitude of the fitted sine is used because the offset is determined by the initial state and therefore by the duration of the resonant pulse. In Fig. 55 the dependence of off-resonant excitations on the angle of the resonant pulse for a rectangular pulse is shown. Performing the measurement for shaped pulses is difficult because a small error in the initialization pulse leads to a non negligible error in the measurement of off-resonant excitations. Additionally the quantum projection noise increases when the state is a mixture of $|0\rangle$ and $|1\rangle$. Therefore the measurement for shaped pulses and different initial states could not be carried out successfully. An attempt of taking a dataset with a Rabi frequency of $\Omega = 2\pi\,160\,\text{kHz}$ and a detuning of $\delta = 2\pi\,200\text{kHz}$ and a Blackman shaped pulse with $T_{slope} = 3\mu\text{s}$ has been made. For most of the collected data it was not possible to fit a sine function properly and therefore the data is not presented here. In section(2.1) the parameter ϵ^2 was introduced as a measure for off-resonant excitations which is does not depend on the initial state. However, it was not possible to measure small off-resonant excitations and therefore it is not possile to calculate ϵ^2 as only an approximation for small ϵ^2 is derived.

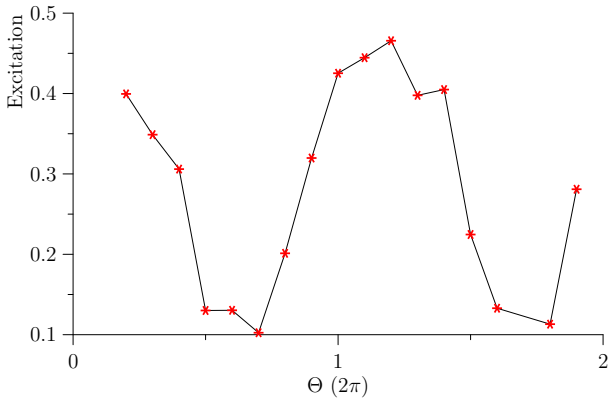

Figure 55: Population oscillation as a function of the angle of the resonant pulse. The parameters are: $\Omega = 340\,\text{kHz}$ $\delta = 400\,\text{kHz}$.

The data presented above shows clearly that off-resonant excitations are understood and that the off-resonant excitations can be minimized by using amplitude shaped pulses.

4.4 CNOT process tomography

Process tomography is a method to fully characterize a quantum process, e.g. a quantum gate operation [9]. A process is a completely positive linear map which describes the dynamics of a quantum system.

$$\rho \rightarrow \epsilon(\rho)$$

This process $\epsilon(\rho)$ may be expressed by an operator sum $\epsilon(\rho) = \sum_i A_i \rho A_i^\dagger$. The operators A_i can themselves be expressed using a fixed set of operators $A_i = \sum_m a_{im} \hat{A}_m$. Therefore, $\epsilon(\rho)$ may be rewritten as

$$\epsilon(\rho) = \sum_{m,n=0}^{4^N-1} \chi_{nm} \hat{A}_m \rho \hat{A}_n^\dagger \tag{18}$$

where N is the number of qubits and $\hat{A}_m$ are operators forming a basis in the space of $2^N \times 2^N$ matrices. The process matrix χ gives a full characterization of the operation. For one qubit operations $\hat{A}_m$ may be the Pauli operators $\hat{I}$, $\hat{\sigma}_x$, $\hat{\sigma}_y$, $\hat{\sigma}_z$, whereas for two qubit operations combinations of these Pauli operators are used. By measuring the final state for a set of 4^N linearly independent input states the process matrix may be reconstructed by inverting relation Eq. 18. Due to uncertainties in the experimental data the results achieved by simply calculating the inverse of Eq. 18 may be unphysical, which means that χ may not be completely positive. Therefore the process matrix is evaluated by performing a maximum likelihood analysis to find the quantum process which fits best to the measured data. The process quality can be characterized by the overlap of the measured χ_{exp} with the ideal process matrix χ_{id}. This measure of quality is usually called the process fidelity

$$F_{proc} = tr\left(\chi_{id} \cdot \chi_{exp}\right)$$

Another characterization of the process quality is the mean fidelity F_{mean} which is calculated by averaging over the measured state fidelity over a large number of input states. More precisely, this involves evaluating $F = \langle \Psi_{out,id} | \, \epsilon(\rho) \, | \Psi_{out,id} \rangle$ for a large number of randomly generated input states and averaging over the results.

In our experiment, process tomography was carried out to characterize the performance of our CNOT implementation [31]. Fig. 56 shows the absolute values of an obtained χ matrix. To investigate the influence of shaped pulse on a complex sequence, a CNOT process tomography was performed with and without shaped pulses. The mean fidelity was increased from $F_{mean} = 84.3\%$ without pulse shaping to $F_{mean} = 89.4\%$ with pulse shaping. With further optimization of the experimental parameters a maximum mean fidelity of $F_{mean} = 92.6\%$ was achieved.

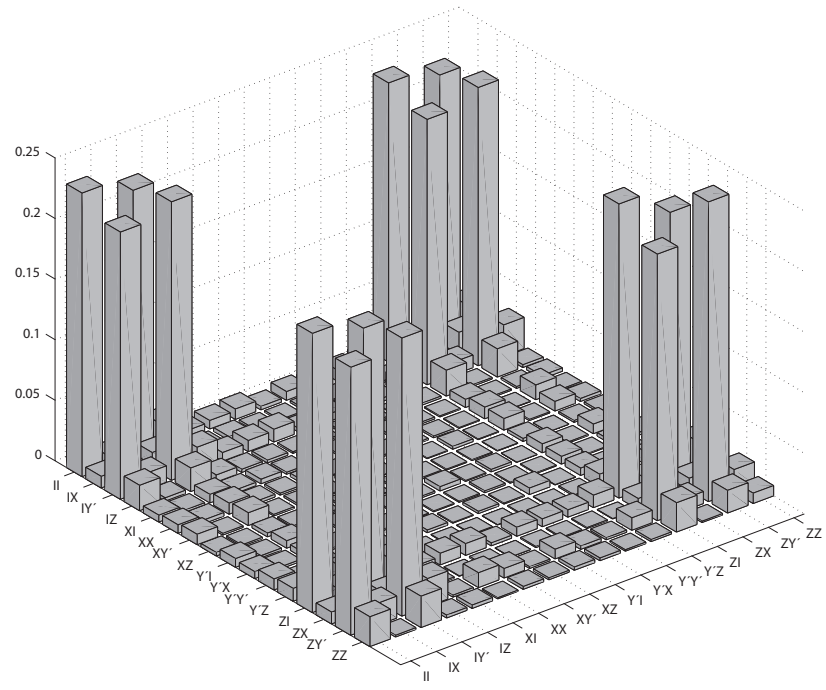

Figure 56: Absolute value of an experimentally obtained χ matrix for a CNOT gate [31].

5 Conclusions and Outlook

The aim of this work was to set up the programmable pulse generator and quantify the influences of amplitude modulated laser pulses on off-resonant excitations. The architecture and capabilities of the programmable pulse generator have been introduced and discussed briefly. Additionally, the integration of the programmable pulse generator in the existing experimental setup has been described. A detailed manual for setting up and programing the programmable pulse generator is given in Appendix (B).

In section (2.1) off-resonant excitations were introduced and it was shown that they can be eliminated by switching the interaction on and off adiabatically. A simple method for testing if a given pulse shape and ramp duration are in the adiabatic regime was given in section (2.2). A Blackman pulse shape was defined and it was shown that this shape with a ramp duration of $5\,\mu$s suppresses the off-resonant excitations on the carrier transition efficiently while driving sideband transitions for the parameters used in our experiment. The theory presented in this thesis was verified in several experiments with rectangular and Blackman shaped pulses. Further, it has been shown that the fidelity of a CNOT gate is increased by applying the pulse shaping techniques described in this paper.

Due to various other improvements in the setup the qubit coherence time and therefore also the gate fidelities will increase in the near future. Another possibility to increase the performance is to generate numerically optimized pulse sequences. One possibility to calculate these optimized sequences is the gradient ascent pulse engineering method [32]. With this method it is possible to optimize the pulse sequences for various criteria. For the application of this technique arbitrary pulse shapes of the amplitude and the phase of the radio frequency pulses are required. Hardware-wise this is already possible with the currently used programmable pulse generator. However the experiment control software has to be almost completely rewritten to offer this flexibility. Another perspective for the future are different type of phase gates like the gate proposed by Mølmer and Sørensen [33]. Since off-resonant excitations are one of the biggest predicted error sources in this type of gate, pulse shaping will be of crucial importance for its implementation.

The development of the programmable pulse generator in the near future will include a new, more flexible radio frequency generation scheme, more digital output channels and a more specialized firmware. The planned radio frequency generation will include a different direct digital synthesizer and an additional FPGA on the same printed circuit board to increase flexibility and to make the generation of arbitrary pulse shapes an easier task for the compiler.

Appendix

A Abbreviations

AOM: acousto optical modulator

API: abstract programming interface

CNOT: controlled not

DAC: digital to analog converter

DDS: direct digital synthesizer

DHCP: dynamic host configuration protocol

FPGA: field programmable gate array

I2C: inter-integrated circuit serial bus

IP: Internet protocol

LVDS: low voltage differential signaling

MAC: media access control

NMR: nuclear magnetic resonance

OSI: open systems interconnection

PCP: pulse control processor

PCB: printed circuit board

PLL: phase locked loop

PPG: programmable pulse generator

PTP: pulse transfer protocol

RAM: random access memory

RF: radio frequency

SFDR: spurious free dynamic range

TTL: transistor transistor logic

TCP: Transmission control protocol

UDP: user Datagram protocol

VCO: voltage controlled oscillator

VHDL: VHSIC hardware description language

VHSIC: Very-High-Speed Integrated Circuits

VGA: variable gain amplifier

B Programmable pulse generator manual

B.1 The python server

In this section the commands for programming the programmable pulse generator (PPG) are explained and general instructions for using the PPG are given. A more recent version of this documentation may be found on the pulse sequencer homepage[25].

Typographic conventions Python code is written as :

```
python_function(arg1,arg2=10)
python_var=True
```

Filenames and command line commands are written as: `path_to/filename.txt` .

B.1.1 Installing the python server

As a prerequisite the python programming language in version 2.4 or 2.5 is required. It may be downloaded from

`http://www.python.org` .

The python server source code for generating and executing pulse sequences for the PPG are available for download at the PPG project homepage.

`http://pulse-sequencer.sf.net`

The package for QFP2.0 is called `sequencer-python/python-qfp-2.0`

The package for QFP_LIN is called `sequencer-python/python-qfp-2.0`

B.1.2 starting up the python server

The server may be started directly by the command line if the python executable is within the search path of the environment variable.

`python start_box_server.py`

The command line options for this command are:

`--debug debug_level` Where the debug levels are supposed to be used as follows:

- 1 : The most important server messages are displayed
- 2 : Many server messages are displayed
- 3 : Many compiler messages are displayed. It's not recommended to use this with the server. This debug level is intended for low level debugging in the machine code generation code

`--nonet` The python server is started without any network connection. This switch may be used for testing the server on a computer where no PPG is connected.

[25] http://pulse-sequencer.sf.net

B.1.3 Troubleshooting

Error messages from the server are generally displayed on the command line terminal executing the server. If the command line terminates without displaying an error messages, open a new command window and run the server in this window. This command window does not close when the python server crashes. If the box is connected to the voltage supply the LEDs beneath the DIP switch for selecting the MAC address should blink a few times while booting up. If this is not the case the clock settings and the presence of a clock signal should be checked. Below a few error messages and possible solutions are described.

No pulse transfer protocol reply: If the server complains that it got no pulse transfer protocol reply there may be a hardware or network problem. Following steps are recommended for troubleshooting:

- If it worked before it may help to flush the ARP[26] cache of windows. Open a command shell and type

`netsh interface ip delete arpcache` . If this does not solve the problem, waiting for 10 minutes before retrying to start the server may be the solution.

- The network connectivity can be checked with "link" LEDs at the FPGA board. If this LED is dark, it's most probably a hardware problem. It should be checked if the network cables are plugged in correctly and the right cable type (No crossed connections) is used.

- The DIP switch for setting the MAC address of the FPGA board might not be set to match the settings given in the python compiler configuration. See the section about configuring the server for details.

- There is no DHCP server running on the network where the box can get an IP address from or the box may be misconfigured so it doesn't send an DHCP request. The DHCP server is most likely implemented in an Ethernet router.

For further troubleshooting it is advisable to use a network analyzing software like the freely available wireshark[27].

When investigating the network traffic with wireshark, restart the FPGA and look for a MAC address of 00:01:ca:22:22:xx performing a DHCP request (xx is the value of the DIP switch for setting the IP address). If this request is accepted, the FPGA is obtaining a IP address successfully. It is likely that the error source is then a misconfigured python server.

[26] address resolution protocol
[27] www.wireshark.org

KeyError , AttributeError If the server returns some strange errors like KeyError or AttributeError there might have been an error with transferring the variables from LabView to the server. Another possibility is that the current sequence is using a TTL channel which is not defined in the hardware configuration file. Check if all TTL channels used in the current sequence are available in the QFP2.0 hardware configuration file.

Pulses still overlap If you got this error and the sequential mode is used there might be something wrong with the delay times in `Innsbruck/__init__.py` . If this error occurs while running in a parallel environment t some overlapping pulses may have been defined. This might not be a problem but the timing of your script may be incorrect.

Other errors One error that occurs frequently in new installed systems are incorrect decimal separators sent from LabView to the python server. Check that the decimal separator is a point "." and not a comma "," .

B.1.4 Pulse programming reference

There are two fundamental modes of programming a pulse sequence:

- sequential programming (default, used for QFP2.0)
- parallel environments (used for QFP Lin)

B.1.5 Sequential programming

This is the default programming mode where the pulses are usually executed sub sequentially. Delays between pulses may be inserted manually. The structure of a program is as follows:

```
pulse1
pulse2
wait(10)
pulse3
```

This is intended to be used for sequences when the 729 beam is used to make rotations on the qubits. The pulses are characterized by the ion, transition , the duration given in angle, and the phase:

`R729(ion,theta,phi,[transition],[start_time],[is_last])`

If the transition is omitted a default transition given by the Labview control program is used.

The command for inserting a waiting time is.

`seq_wait(time)`

The pseudo XML pulse program structure A sequence file is read by the LabView program as well as by the python server. Therefore more information than just the pulse sequence is stored in one file. This is realized by creating groups of information

which are separated by XML[28] tags. The tags which are interpreted by the python server are <VARIABLES>, <TRANSITION> and <SEQUENCE>. In the <VARIABLES> group the variables which are controlled by and transmitted from LabView are defined. In the <TRANSITION> group the transition objects which are defined in LabView may be edited before the phase accumulators in the FPGA are initialized. The <SEQUENCE> group contains the actual sequence.

```
<VARIABLES>
Duration=self.set_variable("float","Duration",20,1,2e7)
</VARIABLES>
<some_tag_for_labview>
some content for labview
</some_tag_for_labview>
<SEQUENCE>
TTL("PM_trigger",50)
seq_wait(Duration*1000.0)
TTL("PM_trigger",50)
</SEQUENCE>
```

Variable definition The variable definition in the XML file has the following syntax:

```
VAR_NAME=self.set_variable("TYPE","LV_NAME",[MIN],[MAX])
```

VAR_NAME:
: The name of the python variable which is used in the script

`TYPE:`
: The type of the variable. One of FLOAT, INT, BOOL, STRING

`LV_NAME:`
: The name of the variable in the LabView program.

`MIN ; MAX`
: The bounds of possible values of the variable. This value is only used by LabView and has no influence on the python server.

Pulse code The pulse code is in the `<SEQUENCE>` group. Variables defined as described above may be used as a simple python variable. Below an overview over the different commands is given.

TTL Outputs The names of the TTL outputs are taken from the "Hardware Settings.txt" file of the QFP. If the device is !PB the output is inverted that means that setting the output to 0 will result in a voltage of 3.3V at the according output. The location of the Hardware settings file is determined in `innsbruck/__init__.py` .

```
866 sw.ch=0
866 sw.Device=!PB
```

Generates the device "866 main" . To generate a pulse on this channel the following code is used:

[28] Extended Markup Language

```
ttl_pulse("866_sw",10)
```

This creates a TTL **pulse** of given duration. To switch the state of the TTL outputs on a given position in the sequential environment the `set_TTL` command is used:

```
set_TTL("866_sw",1)
...
set_TTL("866_sw",0)
```

RF pulses The command for generating RF pulses is:

`R729(ion,theta,phi,[transition],[start_time],[is_last])`

The technical details of the rotation are given by the transition object. See the paragraph above how to define these transitions.

Depending on the version of the python compiler used the output for the second DDS does **not** support phase coherent switching. The actual implementation of this command may differ from one particular experiment to another. It is advisable to customize the pulses for the particular experiment with the help of include files as described below.

Defining Transitions Transition objects in the python server are needed to do phase coherent switching between pulses. A transition object has information about the frequency, the pulse shape, the amplitude and the Rabi times of the radio frequency pulse which is needed to excite the given atomic transition. Transitions may be defined in two different ways:

- Directly in the sequence file
- From LabView via the TRANSITION keyword

An example for the direct definition

```
Carrier=transition(transition_name="Carrier",t_rabi=t_carr,
frequency=freq1,amplitude=1,slope_type="blackman",
slope_duration=0.2,amplitude2=-1,frequency2=0)
```

This transition will be called by `R729(ion,theta,phi,Carrier)`
A transition named carrier which is transmitted from LabView is called by: `R729(ion,theta,phi,'carrier')`

Note that the name of the transition is now a string, whereas by direct definition above the transition object is handled as a variable. The information of the transition is submitted from LabView to the server.

seq_wait The syntax for the sequential wait function is:

```
seq_wait(time)
```

Where time is the waiting time in us. The specialty of this function is that it generates pre and post delays depending on the previous and following commands to ensure that the pause has the expected length. For an example if a seq_wait() instruction is in between to

R729() commands, it will count the delay between the two points where the amplitude of the slopes are at the half maximum. An error message will be returned if the wait duration is smaller than the slope duration plus the frequency switching delays.

The is_last variable There is a possibility to generate overlapping pulses by using the `is_last` variable. For a sequence which generates the following pulses

```
TTL "866_sw" from 0 to 100
TTL "397_sw" from 90 to 110
```

This would look in the sequential mode like:

```
ttl_pulse("866_sw",100,start_time=0,is_last=False)
ttl_pulse("397_sw",20,start_time=90)
```

The `is_last` variable is by default `True`.

Defining new commands In the `Innsbruck/__init__.py` file the include directories are defined:

```
includes_dir="e:/My_Documents/qfp_2.0/PulseSequences/Includes/"
```

Writing include files A sample include file may look like:

```
description="long_optical_Pumping_for_ca43"
function_name="ca43.OpticalPumping2"
arguments="optional: length=50"

class OpticalPumping2(PulseCommand):    #for Ca43
  def __init__(self,length=50):
    configuration=self.get_config()
    ttl_pulse(length,"7",is_last=False)
    ttl_pulse(length+1,"8",start_time=0)

ca43.OpticalPumping2=OpticalPumping2
```

This defines the function `ca43.OpticalPumping2` which may be used in a sequence file. The variables description, Function_name and arguments are necessary for the built in documentation system.

The prefixes for Include functions are:

- `ca40` for the linear trap
- `ca43` for the ca43 experiment
- `cqed` for the CQED experiment
- `segtrap` for the segmented trap

To avoid confusion the prefix for the actual experiment should be used.

Documenting Include files The variables `description`, `function_name` and `arguments` are used to automatically generate a documentation of the commands defined in the include files. To display this list the python `script include_doc.py` which resides in the same directory as the `start_server.py` script is used.

B.1.6 Parallel environment

In this programming mode the absolute start times in relation to the start of the sequence and the durations of the pulses are given. The pulses may overlap but they **cannot have the same start or stop time**. An example parallel program:

```
start_parallel_env()
ttl_add_to_parallel_env("866 sw",0,10)
ttl_add_to_parallel_env("397 sw",2,5)
shape_add_to_parallel_env(start_time=12,duration=5.0,frequency=freq1,\
phase=0,type="blackman",slope_duration=1.1,amplitude=1.0)
end_parallel_env(trigger="Line",repeat=30)
```

This switches on the "866 sw" output at 0us for 10 us and the "397 sw" output at 2μs for 5μs. To some extend timing conflicts which arise from overlapping pulses may be resolved. The server sends a warning response if the conflict is not resolved successfully. The last line repeats the sequence 30 times and waits for a line trigger every time it is running a single repetition. A parallel environment is started with the start_parallel_env() function. Prior to this function the coherent frequency initialization may be performed. The end of a parallel environment is set with the end_parallel_env(trigger,repeat) function. The trigger variable is either "None" or "Line". When it is set to "Line" the PPG waits for a rising slope on the Trigger input 0.

Coherent frequency initialization Before using phase coherent frequency switching the frequencies have to be initialized at the beginning of your program :

```
freq1=coherent_create_freq(frequency,0)
first_dds_init_frequency(freq1)
```

TTL pulses A TTL pulse is generated with the command

```
ttl_add_to_parallel_env(ttl_channel,start_time,stop_time)
```

RF pulses RF pulses are generated with the command

```
get_shaped_pulse(duration,frequency,type,[slope_duration=0],\
  [amplitude=1],[phase=0],[frequency2=0],[amplitude2=0])
```

If `amplitude2` is bigger than 0 than the second DDS channel is switched on with frequency `frequency2` . If `slope_duration` is bigger than 0 than the output is a pulse with a Blackman shape where the slope duration is given in microseconds.

It is not advisable to use arbitrary numbers for the slope_duration and amplitude. The program pre compiles the shapes because they take a long time to compile. It has also to recompile if the amplitudes of the pulses change.

B.2 Configuring the software

This section is divided in two parts. In the first part the steps necessary for setting up a "standard" PPG for use with QFP is described whereas the second part covers all configuration possibilities. Any change in the configuration requires a restart of the server.

B.2.1 Basic configuration

This part relies on a working QFP2.0 environment. The configuration files for the python servers reside in the `innsbruck` and the `test_config` directories. The main files are the `__init__.py` files in the respective directory.

In the innsbruck directory the `configuration` class contains the configuration information. To alter the configuration the methods of this class may be changed.

The hardware configuration file The location of the hardware configuration file is defined by the `hardware_config` method of the `configuration` class. This file is generated by the QFP2.0 `Settings Editor`.

The sequences directory The sequence directory is defined by the `sequences_dir` method of the configuration. An example is `sequences_dir=path_to_qfp/PulseSequences/`.

The includes directory The includes directory is defined by the `includes_dir` method of the configuration. An example is `includes_dir=path_to_qfp/PulseSequences/Includes/` .

Setting the MAC address of the FPGA board The address of the FPGA board is set in the `__init__.py` file in the `test_config` directory. The address is set via the DIP switches on the FPGA board. The function of the DIP switches are defined as follows:

Reset	IP1	IP2	IP3	IP4	DHCP

Where the address of the FPGA board is calculated as $IP1 + 2IP2 + 4IP3 + 8IP4$. The address is set in the `mac_byte` variables of the sequencer generator statements in the `__init__.py` file. All occurrences of the sequencer generator have to be altered.

B.2.2 In depth configuration

TCP ports The communication with LabView is handled over the TCP protocol. The server is listening by default on port 8880. The port is set in the `default_port` method of the `configuration` class.

Activating the parallel mode For using with the old qfp the parallel mode has to be activated. This is done by setting the `parallel_mode` method of the `configuration` class to `True`.

Setting the TCP server mode The TCP server is able to switch between different TCP connection protocols. All variables discussed here are methods of the `configuration` class. If the `answer_tcp` method is set to `True`, the server sends a response to the LabView program after finishing compiling and transferring the sequence. If the method `send_pre_return` is `True`, an additional answer is sent to LabView after error checking but before compiling the sequence. This is helpful if the compilation time is long and therefore it is possible to determine whether the server is not responding or it is busy with compiling a sequence. For the standard settings of QFP2.0 these methods have to be set to `True`.

If the `disconnect_tcp` method is set to `True`, the TCP connection is terminated after each LabView command. This may be helpful if problems with a long lasting TCP connection may arise. For a standard QFP2.0 setup this method should be set to `True`.

Resetting the TTL outputs If it is desired to reset the TTL outputs of the PPG with the beginning of each sequence the `reset_ttl` method of the `configuration` class may be set to `True`.

Configuration of pulse shapes Pulse shapes have to be configured in two files in the `innsbruck` directory. First an entry has to be made in the `pulse_dictionary` method of the `configuration` class. In this dictionary a function has to be assigned with a string. The functions for the pulse shapes are defined in the `pulses.py` file.

Calibration of the VGA The calibration function for the DAC and the VGA is defined in the `calibration` function at the end of the `__init__.py` file. For the QFP2.0 environment the input value is the desired output power in dB where 0 is the maximum and the return value is the DAC value which is an integer between 0 and 16383. For the old QFP the input value is a linear power value where 1 corresponds to the old Marconi setting of +13dBm.

Syntax of the hardware configuration file Normally the hardware configuration file is generated by QFP2.0. For testing purposes it might be necessary to generate a different hardware configuration file. The hardware definition syntax is as follows:

```
854 sw.Device=PB
854 sw.ch=15
866 sw.Device=!PB
866 sw.ch=17
```

The `!PB` indicates a inverting channel. It doesn't matter if the device or the channel is the first argument.

B.2.3 Configuring the devices

The compiler has to be reconfigured if you make changes to your hardware settings; e.g. adding an additional DDS , ...

Configuring the devices For configuring the devices it may be necessary to edit the `innsbruck/__init__,py` as well as the `test_config/__init__.py` files

The timing and the RF frequency generation depend on the clock of the DDS boards and the FPGA. It is assumed that the FPGA clock is derived from the DDS clock and that it operates at 1/8 of the DDS clock frequency.

```
ref_freq=800
cycle_time=(1e3/ref_freq)
```

The cycle time is given in nanoseconds. Every other time used within the compiler should be given in microseconds.

```
ttl_device={}
```

The ttl_device is a dictionary where the corresponding channels to the hardware file are stored. It also stores whether the channel is inverting.

```
first_dac_device=dac_factory.create_device(chain_address = 0x01)
```

This defines the first DAC which is accessible via innsbruck.first_dac_device. If another DAC should be added just another line which is similar to this and a command in the `api.py` file is necessary .

```
dds_factory_create_devices(chain_address={1; 0x1, 2; 0x2},ref_freq=ref_freq)
```

This defines the first DDS device. Note that it's not possible to mix up DDS with different reference frequency because the data transfer relies on fixed reference frequency dividers between the FPGA clock and the DDS clock.

B.3 Configuring the Hardware

Details on the firmware and on programming the FPGA are given in section(C). In this section the configuration of the FPGA board, the chain boards and the evaluation boards is covered

The FPGA board The jumpers CLK0 and CLK2 select the different clock options. Where internally CLK0 is routed to the PCP core clock and CLK2 is routed to the PTP and the Ethernet clock.

The DIP switch J3 determines the network address and the DHCP mode the switch is configured as:

Reset	IP1	IP2	IP3	IP4	DHCP

Where the address of the FPGA board is calculated as $ADDR = IP1 + 2IP2 + 4IP3 + 8IP4$. The other switches are not used in the current configuration. The ip address is 192.168.0.220 + ADDR and the MAC address is 00:01:ca:22:22:ADDR .

The chain boards The chain boards for the DAC and the DDS are configured by a DIP switch. The pin configuration is as follows:

Addr0	Addr1	Addr2	Addr3

The address is calculated as $Addr0 + 2\,Addr1 + 4\,Addr2 + 8\,Addr3$.

The DDS evaluation board On the dds evaluation board the parallel board connector U4 has to be removed. The jumper W2 has to be set to external mode.

The DAC evaluation board The output transformer T1 has to be removed and the solder bridge JP8 has to be closed. The DAC clock has to be controlled externally, therefore the jumper JP2 has to be set and the solder bridge JP3 has to be closed.

B.4 The LabView interface

In this section the interface between the python server discussed above and the experiment control software is described. As for the programming of the sequence there exist two different modes:

- Parallel environment: Build to be compatible with the old qfp program and using the Matlab sequence files
- Sequential environment: Works with the new qfp program and uses python sequence files

The communication is based on a plain text transmission via a TCP socket, where the listening program (server) is the python environment and the sending program is the LabView experiment control program. The standard TCP port is 8880.

B.4.1 Parallel environment

In the parallel environment LabView just sends the whole pulse sequence as a string to the server. All TTL and RF pulses are calculated in the LabView program. There are no additional global variables.

B.4.2 Sequential environment

In sequential mode the transmission from LabView to python consists of a command string with information about Rabi times, RF amplitudes and the sequence file Name. The sequence file is then read out and compiled by the python server. The sequence file format is described above.

General format The server accepts strings in the following format:

`command1,option1_1,option1_2;command2,option1,option1`

a simple example:

`NAME,test_ttl.py;TRIGGER,NONE;FLOAT,duration,3.4;`

The available variables are:

Variable name	Description	Usage
NAME	The name of the sequence file	NAME,file_name
INT	Sends an integer value to the script	INT,var_name,value
FLOAT	Sends a float value to the script	FLOAT,var_name,value
STRING:	Sends a string value to the script	STRING,var_name,value
BOOL	Sends a Boolean value to the script	BOOL,var_name, value
TRIGGER	Gives the type of the trigger.	TRIGGER,trig_string

Possible Trigger strings are: NONE , LINE . The commands are defined in the file `innsbruck/handle_commands.py` .

The transition object The transition object is defined as follows:

TRANSITION, transition_name; RABI, Rabi_times;
SLOPE_TYPE, slope_type; SLOPE_DUR, slope_duration;
AMPL, slope_ampl; FREQ, frequency; IONS, ion_map;
AMPL2, second_amplitude; FREQ2, second_frequency;

Where Rabi frequencies for multiple ions are defined as:
1:19.34 , 2:21.12 , 3:20.34 , 4:22.67
And the ion_map:
1:101 , 2:102 , 3:103 , 4:104

The default transition There is the possibility to define a default transition which is used if no transition is given in the R729 command:

DEFAULT_TRANSITION,transition_name;

B.4.3 Creating pulse commands

The functions intended for use in user level are defined in `user_function.py` or in `api.py` . There exists a framework for handling global variables in a sequence and transferring this variable back to LabView.

An example user mode function is the PMT Detection function:

```
class PMDetection(PulseCommand):
  def __init__(self,detect_wait,CameraOn=False):
    configuration=self.get_config()
    detection397=configuration.detection397
    PMTrigger=configuration.ion_trig_device
    PMGate=configuration.PMGate
    trigger_length=configuration.PMT_trigger_length
    detection_count=self.get_variable("detection_count")
```

```
self.set_variable("detection_count",detection_count+2)
self.add_to_return_list("PM_Count","detection_count")
seq=TTL_sequence()
ttl_pulse(detection397,detect_wait,is_last=False)
ttl_pulse(PMGate,detect_wait,is_last=False)
seq.add_pulse(PMTrigger,trigger_length,is_last=False)
if CameraOn:
  ttl_pulse(trigger_length,configuration.Detection,is_last=False)
  ttl_pulse(trigger_length,configuration.CameraTrigger,is_last=False)
  ttl_pulse(PMTrigger,trigger_length,
      start_time=detect_wait-trigger_length,is_last=False)
```

This function can be divided in following parts:

- class definition
- variable definition from the configuration in `__init__.py`
- handling of the return variables
- the main ttl sequence

This function generates the TTL pulses required for a detection sequence. It generates different pulses if a CCD camera is present. Additionally it uses the variable "detection_count" to send information about the number of detection pulses to LabView.

class definition for user level functions The class definition has to refer to the parent class (PulseCommand) and the method (function) that is executed when the class is called is __init__.

```
class PMDetection(PulseCommand):
  def __init__(self,detect_wait,CameraOn=False):
```

The PulseCommand class contains methods to handle local variables. A sample code that increases a variable each time a command is executed and returns this variable to LabView may look like:

```
class counter1(PulseCommand):
  def __init__(self,value=0):
    self.value=value
  def incme(self):
    self.value+=self.increase
    def return_to_labview(self):
    self.set_variable("counter",self.value)
    self.add_to_return_list("COUNTER_1","counter")
```

it is used in python like:

```
inc1=counter1()
inc1.incme()
...
```

```
...
if CameraOn
  inc1.incme()
...
...
inc1.return_to_labview()
```

B.4.4 Returning values to LabView and using global variables

To return a value to LabView you have to create a global variable:

```
detection_count=self.get_variable("detection_count")
self.set_variable("detection_count",detection_count+1)
self.add_to_return_list("PM_Count","detection_count")
```

First the current value of the variable is obtained. Then the variable is increased and returned back to the global variable. Then the global variable is added to the return list. If the variable is a list, the returned values are separated by commas.

B.4.5 User function framework

The supplied methods for user functions are

```
self.get_config()
```

Returns the global configuration class which is defined in innsbruck/__init__.py

```
self.set_variable(variable_name,value)
```

Sets a global variable. You can assign all types **except** dictionaries !!!

```
self.get_variable(variable_name,[default_value])
```

Returns the value set by set_variable().

If the variable is not set yet it returns default_value. If default_value is omitted the default_value is 0.

```
self.add_to_return_list(Pre_String,variable_name)
```

Adds the variable to the LabView return list. The variable is returned as:

`"Pre_String,value;"`

or if the variable is a **list** [var1,var2,var3,...]:

`"Pre_String,var1,var2,var3,...;"`

```
self.get_error_handler()
```

Returns the error handler function to return error messages to LabView.

```
self.get_cycle_time()
```

Returns the clock cycle time in microseconds.

```
self.address_ion(ion,[start_time])
```

Checks if the current ion is the parameter ion. If the ion has to be changed it invokes self.ion_event(). It handles the ion return list as well

`self.ion_event()`

A pointer to the TTL event class which switches the ions. Normally this is parallel_ttl(); Within the TTL_sequence class this is the method self.add_pulse.

B.5 Internals of the Software

In this section some details of the python compiler are described. For a general overview see section(3.4).

File locations

The directory tree of the compiler is divided in the following directories

`sequencer`	Common parts of the compiler
`sequencer/pcp`	Definition and generation of the pcp commands
`sequencer/pcp/machines`	Implementation and generation of the instructions
`sequencer/pcp/events`	Abstract definition of the events called by the API
`sequencer/pcp/instructions`	Abstract definition of the pcp32 instructions
`sequencer/devices`	Definition of the hardware devices (DDS,DAC)
`sequencer/ptp/`	Definition and implementation of the PTP protocol
`test_config`	Definition of the API and configuration (for all experiments)
`innsbruck`	Definition of additional API and special configuration

The object structure

The compiler is based on an object oriented structure where the main objects are:

sequencer	The sequencer main object.
test_config	Includes the DDS and DAC and TTL objects.
innsbruck	Consists of additional API commands and the end user layer.

Overview

The sequencer object

The most important methods of the sequencer object are:

current_sequence	Methods and variables for managing the array of abstract events.
standard_params	Common constants and definitions.
main_program	Array for the end user layer events

The test_config object

first_sequencer	Object for the FPGA board with ptp address 1. Includes methods for compiling the sequence and sending it over PTP.
first_sequencer.machine	The abstract machine object. Includes methods for compiling the events to machine code.
first_dac_device	Abstract hardware object for the DAC with chain board address 1
second_dac_device	Abstract hardware object for the DAC with chain board address 2
dds_devices	Array of abstract hardware objects for the DDS. The index corresponds to the chain boards address

The innsbruck object

start_server()	Method to initialize the server and the compiler.
error_handler()	Method for returning error messages to Labview

C Internals of the Programmable pulse generator

In this section not all aspects of the programmable pulse generators are covered. For a complete description of the concepts of the programmable pulse generator the reader is referred to reference [25]. In the following hexadecimals values are displayed as 0x(value). For example 0xFF is 255 in the decimal representation.

C.1 The firmware

Compiling the firmware

The firmware is described in the hardware description language VHDL. The source code files available on the project homepage are macro files which itself generate the VHDL source files. This additional step has the advantage that if changes on an object is made, this changes are made in all VHDL files containing this object. For compilation of the VHDL sources the Quartus II design suite available on the Altera homepage[29] is required. The source code may be obtained from the CVS repository of the pulse sequencer project homepage[30]. Additionally the GNU make tools and the M4 scripting interpreter are required. For the windows operating system these are provided by the cygwin[31] environment. To generate the Quartus II project file open a command window and type `make sequencer_top.vhd` and `make sequencer_top.map.eqn`. This creates a file called sequencer_top.qpf which could be loaded in Quartus II and compiled by pressing ctr+L .

[29] www.Altera.com
[30] htttp://pulse-sequencer.sf.net
[31] http://www.cygwin.com

Programming the FPGA The FPGA may be programmed with the programming software provided by the Quartus II design suite. The cable should be connected to the config port on the FPGA port. In the programming software choose the ByteBlaster II cable and the active serial programming mode. The file to program is `sequencer_top.qpf` .

Overview over the firmware

A general overview of the programmable pulse generator is given in section(3.4.3). The firmware consists of different blocks which are shown in Fig. 57. These blocks are interconnected with a system-on-a-chip bus. The "Wishbone" bus standard as defined by the Opencores[32] community is used. The bus system is described in detail in the Master's thesis of Paul Pham [25].

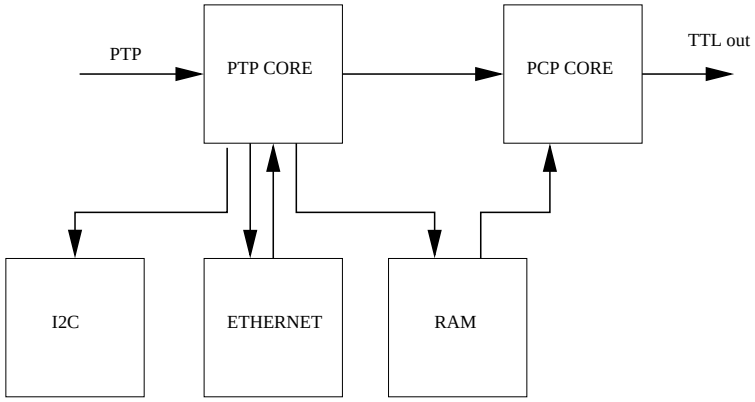

Figure 57: Overview over the firmware of the programmable pulse generator

The network communication

The programmable pulse generator has two different possibilities for communication:

- The Network Stack for communicating with the experimental control computer
- The Daisy-chain Stack for communicating with other programmable pulse generators.

The Daisy-chain stack is only required if two ore more programmable pulse generators are synchronized and programmed by the same experiment control computer. In our setup this feature is not used. Both stacks comply to the standard OSI (Open Systems Interconnection) layer model which are shown in Tab. 10. The layers of the Network Stack are the standard Internet layers as used by normal personal computers. (Ethernet PHY, RJ45) The Daisy-chain Stack uses the same connectors and cabling (RJ45 connector) but utilizes LVDS logic. In difference to the Ethernet specification, the Daisy-chain physical layer is half-duplex to make clock recovery and synchronization easier. The Datalink layer is described by the pulse transfer protocol (PTP) frame definition. A PTP frame is shown in Tab. 11. It consists of fields containing the source and destination addresses, the firmware

[32] http://www.opencores.org

version, the PTP opcode, and the payload. The Network layer handles the routing of this PTP package depending on the values in the `Destination ID.` field. The highest layer is the PTP server which interprets the PTP Opcode and communicates with the other blocks in the programmable pulse generator. The possible Opcodes are shown in Tab. 12. The experiment control computer sends a PTP frame to the programmable pulse generator over the UDP connection. These packets are either routed to other programmable pulse generators in the PTP chain or interpreted by the PTP server. The PTP protocol is described in more detail in the Master's thesis of Paul Pham [25].

OSI Layer	Function	Network Stack	Daisy-chain Stack
Physical	Connector interface	Ethernet PHY, RJ45	LVDS, RJ45
Datalink	Logical Link	Ethernet MAC	Daisy chain link
Network	Global Addressing	IP	PTP routing
Transport	Multiplexing	UDP, TCP	None
Application	API for high level functions	DHCP, PTP server	PTP server

Table 10: The OSI layers of the network communication methods.

Field	Source ID	Dest. ID	Major Ver.	Minor Ver.	PTP Op-code	Zero	Total Length	Unused	Payload
Octets	1	1	1	1	1	1	2	2	variable
Address	0x0	0x1	0x2	0x3	0x4	0x5	0x6	0x8	0xA

Table 11: A pulse transfer protocol frame.

Opcode	Name	Function
0x00	Null	The PTP packet is ignored.
0x01	Status Request	The PPG sends a standard reply to the source of the package.
0x11	Status Reply	The answer to a Status Request.
0x02	Memory Request	Requests a RAM read or write function. The payload contains the data to be written.
0x12	Memory Reply	Answer to Memory Request. The payload contains either a write success word or the data to be read.
0x04	Start Request	Starts or Stops the various blocks of the PPG.
0x14	Start Reply	The answer to a Start Request.

Table 12: The pulse transfer protocol Opcodes.

The PCP Core

The pulse control processor (PCP) core reads the instruction words from the RAM, interprets it and controls the outputs of the FPGA. In Paul Pham's Master's thesis the PCP0 instruction set is described while in the current setup the PCP32 instruction set is used. The main difference is the width of the instruction word. A PCP0 instruction word is 64 bit wide whereas the PCP32 instruction set is 32 bit wide. The PCP0 instruction set has also no support for phase coherent switching. A block diagram of the PCP core is shown in Fig. 58. The Decoder interprets the instruction word and generates control signals for the other blocks. The Program Counter keeps track of the current RAM address and handles program control flow instructions, and the Phase Registers are needed for phase coherent switching. The opcodes for the PCP32 are shown in Tab. 13. The Opcodes for the PCP32 core are documented below. The 4 most significant bits of the 32 bit instruction word are reserved for the opcode. The remaining 28 bits are referred to as [27:0]. (If a function uses the lowest 8 bits it is denoted as [7:0])

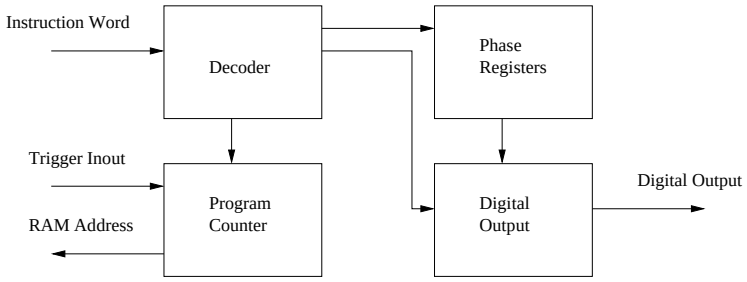

Figure 58: Block diagram of the PCP32 core.

Instruction	Opcode	Description
nop	0x0	No operation
btr	0x3	Branch on trigger
jump	0x4	Jump to address
call	0x5	Subroutine Call
return	0x6	Subroutine return
halt	0x8	Halt pcp
wait	0x9	Wait
bdec	0xA	Branch decrement
ldc	0xB	Load constant
p	0xC	Pulse immediate
pp	0xD	Pulse phase
lp	0xE	Load phase

Table 13: Instruction set of the pcp32

nop

Opcode: 0x0

Name: No Operation

Function: Does nothing

Parameter: None

btr

Opcode: 0x3

Name: Branch on trigger

Functions: Jumps to the address given if the Trigger input matches the given data.

Parameter: Trigger word [27:19]

Parameter: Address [18:0]

jump

Opcode: 0x4

Name: Jump

Functions: Jumps to the given address.

Parameter: Address [18:0]

call

Opcode: 0x5

Name: Call Subroutine

Functions: Calls a subroutine at a given address.

Parameters: Address [18:0]

return

Opcode: 0x6

Name: Return Subroutine

Functions: Returns from a subroutine to the former address.

Parameters: None

halt

Opcode: 0x8

Name: Halt PCP

Functions: Stops the PCP core.

Parameters: None

wait

Opcode: 0x9

Name: Wait

Functions: Waits for a given number of clock cycles

Parameters: Wait cycles [27:0]

bdec

Opcode: 0xA

Name: Branch decrement

Functions: Decrements the loop register and branches to the given address if the value in the register is bigger than zero.

Parameters: Address [18:0]

Parameters: Register Address [27:23]

The PCP32 has 64 registers for different finite loops. The Register Address word selects the register.

ldc

Opcode: 0xB

Name: Load constant

Functions: Loads a constant to the loop register.

Parameters: Constant [7:0]

Parameters: Register Address [27:23]

The PCP32 has 64 registers for different finite loops. The Register Address word selects the register.

p

Opcode: 0xC

Name: Pulse immediate

Functions: Sets the value of the given part of the output registers to the given value

Parameters: Value [15:0]

Parameters: Output select [17:16]

The output select selects the part of the 64 bits of the LVDS output which is controlled. If the value is 0 then the bits [0:15] are set if it is 1 then [16:31] are set, etc.

pp

Opcode: 0xD

Name: Pulse Phase

Functions: Adds the phase offset to the current value of the addressed phase accumulator and sets the corresponding output bits.

Parameter: Phase Offset [15:0]

Parameter: Byte select [16]

Parameter: Current [20]

Parameter: Phase Addend [21]

Parameter: Phase register address [27:23]

As the data bus for the DDS is only 8 bits wide, the 12 bit phase word has to be written in two different write cycles. The Byte select bit selects if the lower 8 or the upper 4 bits are written to the DDS.

lp

Opcode: 0xE

Name: Load Phase

Functions: Sets the value of the frequency tuning word of the given phase register.

Parameter: Phase Value [15:0]

Parameter: Byte select [16]

Parameter: Phase wren [22]

Parameter: Phase register address [27:23]

As the frequency tuning word is 32 bits wide it has to be split up in two 16 bit words.

C.2 Pin configuration of the LVDS Bus

In Tab. 14 the pin configuration of the LVDS bus in the actual setup is shown. The digital output form pin 0 is not used. Also the digital outputs ranging from pin 32 to pin 47 are not used in the actual setup due to firmware limitations.

Pin	Function	Device	Description
0	Digital Out		Not used
1	WRB	DAC	Write enable pin of the DAC
2-15	D0 - D13	DAC	Data bus of the DAC
16	PSEN	DDS	Profile enable pin of the DDS
17	WRB	DDS	Write enable pin of the DDS
18-23	A0 - A5	DDS	Address bus of the DDS
24-31	D0 - D7	DDS	Data bus of the DDS
32 - 47	Digital Out		Digital Out to front panel
48	IO Update	DDS	Register update pin of the DDS
49,50	PS1, PS0	DDS	Profile select pins of the DDS
51-54	BA0 - BA3	DDS chainboard	Chainboard address of the DDS
55-59	Digital Out		Not used
60-63	BA0 - BA3	DAC chainboard	Chainboard address of the DAC

Table 14: Output pin configuration of the FPGA

D Matlab source code

D.1 Simulation of a 2 level system

The following code is a Matlab script to simulate an arbitrary pulse form on a two level system. The concept is described in section(2.3).

```
N0=5000;
t_top=10e-6;
t_slope=4e-6;
w0=1/1.62e-6*2*pi;% The Rabi frequency
delta=600e3*2*pi;% The detuning
beta_array=[];%initialize the arrays used
w_int=0;
result=[];
result1=[];
t_all=2*t_slope+t_top;
t_all_list=[t_all_list,t_all];
%generate the pulse
raising_length=floor(t_slope/(t_all)*N0);
t_raising=[1:raising_length]/raising_length;
w_top=w0*ones(floor(t_top/t_all*N0),1);
w_raising=w0*1/2*(1-cos(t_raising*pi));
w_falling=w0*1/2*(1-cos(pi+t_raising*pi));
omega_t=[w_raising,w_top',w_falling];
t_step=t_all/length(omega_t);
%define the initial state
x1=[1;0];
%do the first evolution step outside the loop
w1=sqrt(omega_t(1)^2+delta^2);
beta=acos(delta/w1)/2;
x2=[cos(beta),-sin(beta);sin(beta),cos(beta)]*x1;
x21=x2;
x31=[exp(i*w1*t_step/2),0;0,exp(-i*w1*t_step/2)]*x21;
x41=[cos(beta),sin(beta);-sin(beta),cos(beta)]*x31;
for (i0=1:length(omega_t))
t=(i0-1)*t_step;
w0=omega_t(i0);
w1=sqrt(w0^2+delta^2);
w_int=w_int+w1*t_step;
beta=acos(delta/w1)/2;
%get the new dressed state coefficient with the new parameters
x21=[cos(beta),-sin(beta);sin(beta),cos(beta)]*x41;
%do the dressed phase evolution
x31=[exp(i*w1*t_step/2),0;0,exp(-i*w1*t_step/2)]*x21;
%rotate back to the unperturbed basis
x41=[cos(beta),sin(beta);-sin(beta),cos(beta)]*x31;
%do the ideal phase evolution (the adiabatic approximation)
```

```
x3=[exp(i*w_int/2),0;0,exp(-i*w_int/2)]*x2;
x4=[cos(beta),sin(beta);-sin(beta),cos(beta)]*x3;
result=[result,x4];
result1=[result1,x41];
end;
figure(1)
plot(abs(result1(2,:)).^2)
figure(2)
plot(t_all_list,abs(x41_result(2,:)).^2)
```

D.2 Calculation of the adiabatic factor

The following code calculates the adiabatic factor α for given detuning and Rabi frequency

```
T2=10e-6
t=[1:1000]/1000*T2;
omega0=461e3*2*pi;
dstart=30;
dstep=30e3;
n0=41
d_array=[];
max_black=[];
max_lin=[];
max_cos=[];
for i0=1:n0
d_array=[d_array,dstart+dstep*i0];
d1=(dstart+dstep*i0)*2*pi;
adia_black = .5*omega0*(sin(t*pi./T2)* pi./T2-.32* ...
...  sin(2*t*pi./T2)*pi./T2)* d1./(d1.^2+.25*omega0.^2 ...
...  *(.84-cos(t*pi./T2)+.16*cos(2*t*pi./T2)).^2).^(3./2) ;
adia_lin = omega0*d1./(T2*(d1.^2+omega0.^2*t.^2./T2.^2).^(3./2));
adia_cos = 4*omega0*sin(t*pi./T2)*pi*d1./(T2* ...
...  (4*d1.^2+omega0.^2*(1-cos(t*pi./T2)).^2).^(3./2));
max_black=[max_black,max(adia_black)];
max_lin=[max_lin,max(adia_lin)];
max_cos=[max_cos,max(adia_cos)];
end;
```

E Bibliography

References

[1] *ARDA Quantum Computing Roadmap*, available at http://qist.lanl.gov (2004).

[2] P. W. Shor, *Algorithms for quantum computation: discrete logarithms and factoring*, Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on pp. 124–134 (1994).

[3] R. Feynman, *Simulating Physics with Computers*, International Journal of Theoretical Physics **21**, 467 (1982).

[4] D. Porras and J. I. Cirac, *Effective Quantum Spin Systems with Trapped Ions*, Physical Review Letters **92** (2004).

[5] T. Schaetz, D. Leibfried, J. Chiaverini, M. D. Barrett, J. Britton, B. Demarco, W. M. Itano, J. D. Jost, C. Langer, and D. J. Wineland, *Towards a scalable quantum computer/simulator based on trapped ions*, Applied Physics B **79**, 979 (2004).

[6] D. P. DiVincenzo, *The Physical Implementation of Quantum Computation*, Fortschritte der Physik **48**, 771 (2000).

[7] C. Monroe, *Quantum information processing with atoms and photons*, Nature **416**, 238 (2002).

[8] D. P. DiVincenzo, *Two-bit gates are universal for quantum computation*, Physical Review A **51**, 1015 (1995).

[9] J. F. Poyatos, J. I. Cirac, and P. Zoller, *Complete Characterization of a Quantum Process: The Two-Bit Quantum Gate*, Physical Review Letters **78**, 390 (1997).

[10] E. Knill, *Quantum computing with realistically noisy devices*, Nature **434**, 39 (2005).

[11] P. W. Shor, *Fault-tolerant quantum computation*, Foundations of Computer Science **96** (1996).

[12] D. Kielpinski, C. Monroe, and D. J. Wineland, *Architecture for a large-scale ion-trap quantum computer*, Nature **417**, 709 (2002).

[13] H. C. Naegerl, *Ion Strings for Quantum Computing*, Ph.D. thesis, Universität Innsbruck (1998).

[14] J. I. Cirac and P. Zoller, *Quantum Computations with Cold Trapped Ions*, Physical Review Letters **74**, 4091 (1995).

[15] W. M. Itano, J. C. Bergquist, J. J. Bollinger, J. M. Gilligan, D. J. Heinzen, F. L. Moore, M. G. Raizen, and D. J. Wineland, *Quantum projection noise: Population fluctuations in two-level systems*, Phys. Rev. A **47**, 3554 (1993).

[16] M. Riebe, *Preparation of entangled states and quantum teleportation with atomic qubits*, Ph.D. thesis, Universität Innsbruck (2005).

[17] B. Shore, *The theory of atomic excitation*, vol. 1, Wiley Interscience (1990).

[18] F. J. Harris, *On the Use of Windows for Harmonic Analysis with the Discrete Fourier Transform*, in *PROCEEDINGS OF THE IEEE*, vol. 66 (1978).

[19] S. Gulde, *Experimental Realization of Quantum Gates and the Deutsch-Josza Algorithm with Trapped Ca Atoms*, Ph.D. thesis, Universität Innsbruck (2003).

[20] H. Haeffner, S. Gulde, M. Riebe, G. Lancaster, C. Becher, J. Eschner, F. Schmidt-Kaler, and R. Blatt, *Precision Measurement and Compensation of Optical Stark Shifts for an Ion-Trap Quantum Processor*, Physical Review Letters **90**, 143602 (2003).

[21] R. W. P. Drever, J. L. Hall, F. V. Kowalski, J. Hough, G. M. Ford, A. J. Munley, and H. Ward, *Laser phase and frequency stabilization using an optical resonator*, Applied Physics B: Lasers and Optics **31**, 97 (1983).

[22] M. Chwalla, K. Kim, T. Monz, P. Schindler, M. Riebe, C. F. Roos, and R. Blatt, *Precision spectroscopy with two correlated atoms*, Applied Physics B: Lasers and Optics **89**, 483 (2007).

[23] F. Schmidt-Kaler, S. Gulde, M. Riebe, T. Deuschle, A. Kreuter, G. Lancaster, C. Becher, J. Eschner, H. Haeffner, and R. Blatt, *The coherence of qubits based on single Ca^+ ions*, Journal of Physics B: Atomic, Molecular and Optical Physics **36**, 623 (2003).

[24] L. S. Ma, P. Jungner, J. Ye, and J. L. Hall, *Delivering the same optical frequency at two places: accurate cancellation of phase noise introduced by an optical fiber over the time-varying path*, Optics Letters **19** (1994).

[25] P. Pham, *A general-purpose pulse sequencer for quantum computing*, Master's thesis, Massachusetts Institute of Technology (2005).

[26] H. Nyquist, *Certain topics in telegraph transmission theory*, Proceedings of the IEEE **90**, 280 (2002).

[27] *Analog Devices: A Technical Tutorial on Digital Signal Synthesis*, available at: http://www.analog.com/dds (1999).

[28] *Analog Devices: AD9858 Datasheet*, available at: http://www.analog.com/dds (2004).

[29] H. Rohde, *Experimente zur Quanteninformationsverarbeitung in einer linearen Ionenfalle*, Ph.D. thesis, Universität Innsbruck (2001).

[30] C. F. Roos, *Controlling the quantum state of trapped ions*, Ph.D. thesis, Universität Innsbruck (2000).

[31] M. Riebe, K. Kim, P. Schindler, T. Monz, P. O. Schmidt, T. K. Körber, W. Hänsel, H. Häffner, C. F. Roos, and R. Blatt, *Process Tomography of Ion Trap Quantum Gates*, Physical Review Letters **97**, 220407 (2006).

[32] N. Khaneja, T. Reiss, C. Kehlet, T. Schulte-Herbrüggen, and S. J. Glaser, *Optimal control of coupled spin dynamics: design of NMR pulse sequences by gradient ascent algorithms.*, J. Magn. Reson. **172**, 296 (2005).

[33] A. Sørensen and K. Mølmer, *Entanglement and quantum computation with ions in thermal motion*, Physical Review A **62**, 022311 (2000).

Danksagung

Diese Arbeit wäre ohne die Hilfe und Unterstützung der gesamten Arbeitsgruppe nicht zustande gekommen. Daher möchte ich mich an dieser Stelle bei ihnen bedanken.

Mein Dank gebührt zuallererst Prof Rainer Blatt der mir mit der Aufnahme in seine Arbeitsgruppe die Möglichkeit gab in einem einzigartigen Umfeld an einem faszinierenden Thema zu arbeiten.

Die Fertigstellung des "Programmable Pulse Generator", der Grundlage dieser Arbeit, wäre ohne die Hilfe von Paul Pham und Timo Körber nicht möglich gewesen.

Die Inbetriebnahme des Experimentes wäre ohne die Hilfe meiner Mitstreiter Mark Riebe, Thomas Monz, Michael Chwalla zu einem Ding der Unmöglichkeit geworden. Ich möchte ihnen für dies und die vielen unterhaltsamen Stunden danken.

Felicity Splatt hat mit wiederholten Korrekturlesen die sprachlich Qualität dieser Arbeit verbessert. Ich möchte mich bei ihr für dies und auch für die vielen amüsanten Unterhaltungen im gemeinsamen Büro bedanken. Für das Korrekturlesen in inhaltlicher Sicht bedanke ich mich bei Wolfgang Hänsel, Markus Hennrich, Timo Körber und Piet Schmidt.

Abseits des Labors möchte ich vor allem meiner Familie danken, die mich immer unterstützten und ermutigten die Arbeit doch endlich zu Ende zu bringen. Weiters bedanke ich mich bei Tanja und allen anderen die mich immer wieder daran erinnern, dass es ein Leben ausserhalb der Universtät gibt.