# Measuring the Molecular Structure of Ultracold Lithium-Rubidium Dimers by Feshbach Resonances

by

Alan Robinson

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

BACHELOR OF SCIENCE

in

The Faculty of Science

(Physics)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

June 2009

# Abstract

Progress has been made towards measuring $^{6}Li^{85}Rb$ Feshbach resonances. These resonances probe the molecular structure of the dimer molecule[2], allowing for future production and spectroscopy of these species. Delays in preparing an absorption imaging setup, lithium magneto-optical trap, and a strong dipole trap have delayed the Feshbach resonance measurement. Progress made in trapping rubidium and imaging of atoms is reported here. In addition, a technical overview of a dual-frequency microwave generator used in the Feshbach resonance experiment is given here.

# Table of Contents

# List of Tables

# List of Figures

# Acknowledgements

I would like to thank

- Dr. Kirk Madison for his enthusiasm and the opportunities he provides.
- Janelle van Dongen for teaching me about the MOT and laser and for the cookies.
- Dr. Bruce Klappauf for looking out for me, and for your software experience.
- Keith Ladouceur for getting me started.
- Dr. Art Mills for sharing his expertise.
- Will Gunton for his help and support.
- and Benjamin Deh for taking over.

# Part I

# Towards $^{6}$Li $^{85}$Rb Feshbach Resonance Spectroscopy

The field of atomic and molecular optics has recently begun exploring the realm of ultracold chemistry. The proliferation of tools for cooling and manipulating atoms since the mid 1990's has allowed the exploration of degenerate gases, cold homonuclear systems, and now ultracold heteronuclear systems. At low temperatures, weakly attractive forces between two alkali species can create bound dimer molecules.[2] The production of ultracold molecules will allow the investigation of many quantum effects. In particular, LiRb dimers hold a large electric dipole moment that can be exploited to produce long range intermolecular interactions. These interactions are critical to future quantum computation schemes.[15]

I had proposed for this thesis to measure the least bound states of the triplet and singlet configurations of $^{6}$Li$^{85}$Rb and $^{6}$Li$^{87}$Rb dimer molecules through Feshbach resonances. The exploitation of Feshbach resonances has allowed the formation of dimer molecules and the mapping of their higher energy bound states.[2] The structure of several heteronuclear dimer molecules of alkali metals have been mapped, including those of $^{6}$Li$^{87}$Rb.[3] These spectra have allowed the determination of the interatomic potentials, and will allow the future development of systems to control the production and manipulation of ultra-cold molecules.[13] While requiring the trapping of two atomic species, these spectra are also fairly easily obtainable.

It has been the difficulties of trapping both lithium and rubidium that has delayed this measurement. Part I will present the steps taken towards taking this measurement as of the end of March 2009.

# Chapter 1

# Theory

## 1.1 The Species

The Quantum Degenerate Gases Laboratory uses a Magneto-Optical Trap capable of trapping $^{6}$Li, $^{85}$Rb, and $^{87}$Rb. Detailed information on the quantum structure of these species can be found at Daniel Steck's Alkali D Line Data website[8, 16, 17]. The structure of the $D_2$ transitions for rubidium are shown in Figure 1.2.

The simple quantum structure of alkali atoms, with a single valence electron, make them simple to manipulate. The electron spin, S, orbital angular momentum, l, and nuclear spin, I, sum to give 3 quantum numbers to access: $F = S + I + l$ - total spin, $m_F$ - the spin projection, and $l$. Changing $l$, by the s-orbital to p-orbital, s-p, transition, requires laser light while changing $F$ in the $l = 0$ state requires radio and microwave radiation.

## 1.2 Magneto-Optical Cooling and Trapping

A magneto-optical trap, MOT, uses the radiation pressure of counter-propagating beams of light to preferentially slow atoms, and to direct them towards the center of an anti-Helmholtz magnetic field as shown in Figure 1.2. To slow atoms, the counterpropagating MOT beams are slightly red detuned from an atomic transition labelled the cycling transition: the $F = 3$ to $F' = 4$ transition in $^{85}$Rb. As the transition probabilities and thus radiation pressure decrease with detuning, the Doppler shift experienced by moving atoms will preferentially slow the atoms and cool the gas.

To keep the atoms confined, an anti-Helmholtz magnetic field is applied to the trap. In this arrangement, the magnetic field points towards the center of the trap on the vertical axis, and away from it on the horizontal plane. By circularly polarizing the MOT beams, $\sigma-$ transitions can be preferentially excited. Due to the positive Landé g-factor for the two MOT pumping states, the transition energy is reduced in regions with larger magnetic fields, reducing the detuning and increasing the radiation pressure. Therefore, a

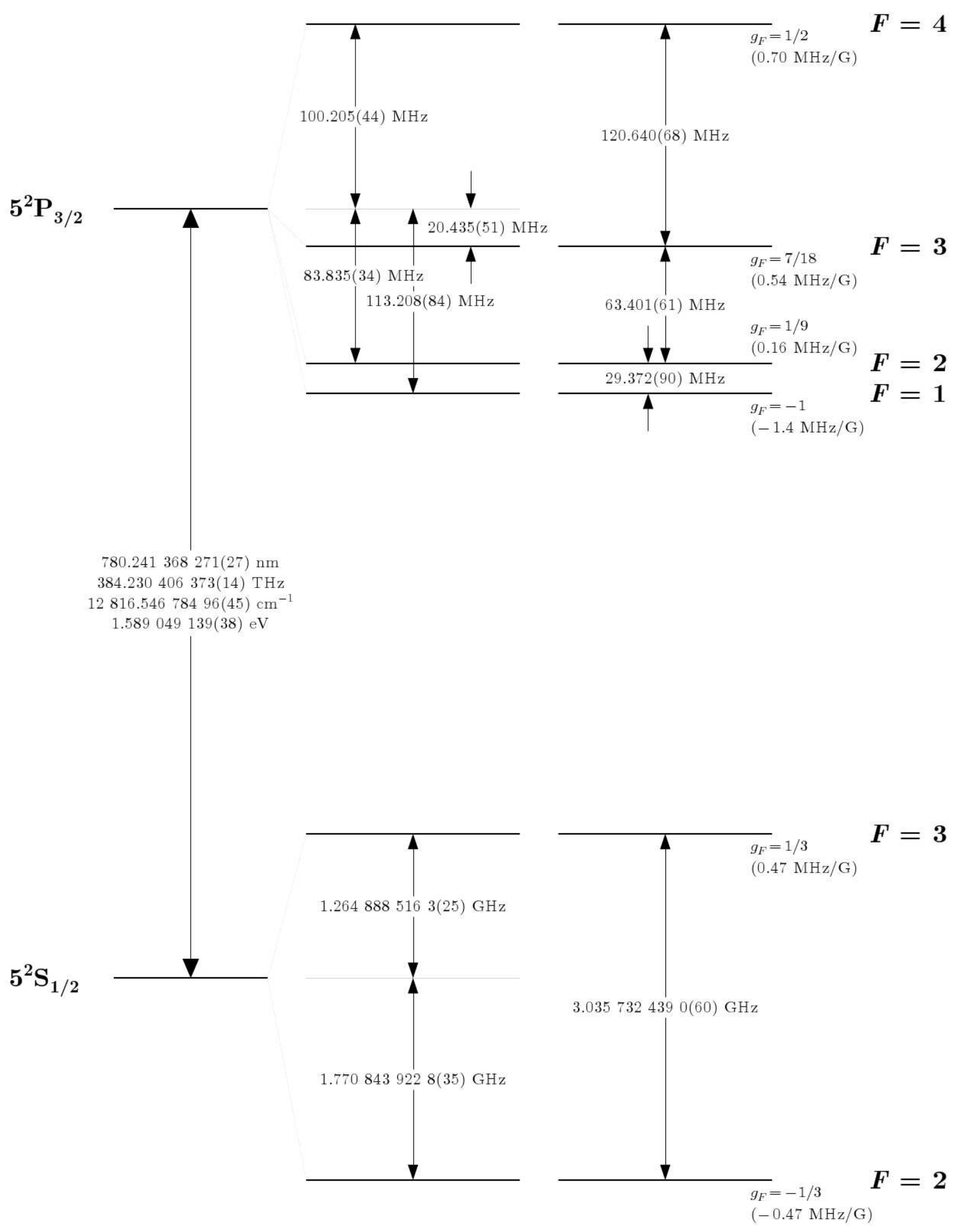


Figure 1.1: Rubidium 85 $D_2$ line data. Figures by Daniel Steck.[17]

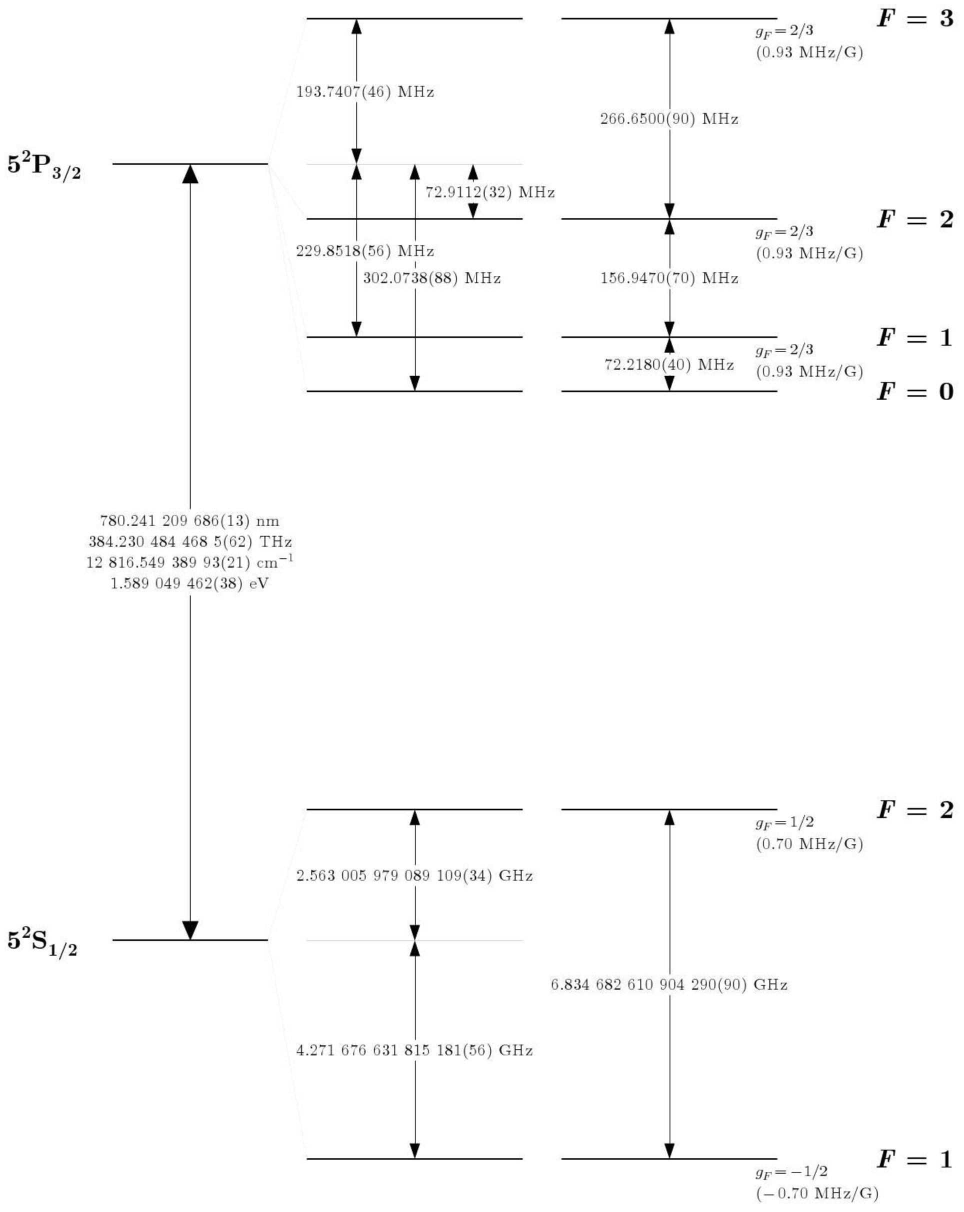
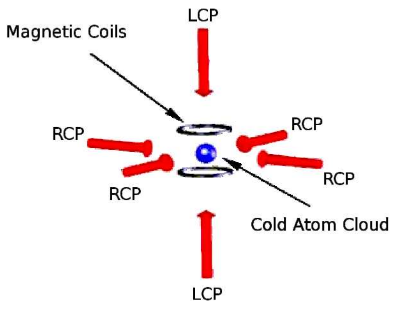
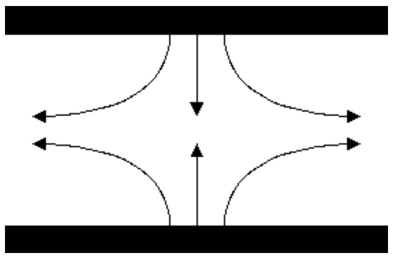


Figure 1.2: Rubidium 87 $D_2$ line data. Figures by Daniel Steck.[16]

preferential radiation pressure is created to direct the atoms towards the zero at the center of the magnetic field.[5]



(a) Physical structure of a magneto-optical trap.



(b) The shape of an anti-Helmholtz magnetic field projected in the vertical plane.

Once cooled, both Li and Rb gases may be trapped together using an optical dipole trap.[5] This trap uses the rapidly oscillating electric field of a tightly focused laser beam and the polarizability of the atoms to confine them. This trap has a very low probability of driving any quantum transitions in the atoms due to the large detuning of the light from these transitions. The tapping potential is of the form $U_{\text{dipole}} \propto \hbar\Omega^2/\delta$ where $\Omega$, the Rabi frequency, is proportional to the beam intensity. The transition probability goes as $1/\delta^2$ and is unimportant for the very large detunings used here. The quantum states of the atoms are preserved in a dipole trap whereas the MOT continually excited the atoms within it.

While a MOT could work with a single light frequency, the pump light, spontaneous decay to both hyperfine ground states requires a second frequency, the repump light, to ensure that all atoms can reach the cycling transition.

## 1.3 State Transitions and Optical Pumping

Atomic quantum states are changed by the absorption or emission of a photon. When resonant light is present, atoms can gain energy, or loose energy through stimulated emission. The rate of absorption and stimulated emission is characterized by the rabi frequency,

$$\Omega = \frac{\vec{D} \cdot \vec{E_o}}{\hbar} \tag{1.1}$$

where $\vec{D}$ is the transition dipole moment. A pulse of radiation tuned to an atomic transition will drive Rabi oscillations between the two states where the populations between the two states oscillate at $\Omega$.[5] When on resonance, a pulse of radiation can be timed to drive half a Rabi oscillation, transferring the population. This is a $\pi$-pulse. During a long pulse, the Rabi oscillations will decohere, producing a mixture of atomic states.

Atoms may also decay spontaneously by any allowed atomic transition. These transitions have a lifetime of 26.25ns for the rubidium $D_2$ line[17], but are unimportant for the hyperfine transitions as the decay rate scales as $\omega^3_{\text{transition}}$. For all operations here, spontaneous emission of the $D_2$ transition is more important than stimulated emission.

The allowed electric dipole atomic transitions for rubidium require $\Delta n \geq 1$ and $\Delta l = \pm 1$, allowing the $D_2$ transitions, and $\Delta F \in \{-1, 0, 1\}$ and $\Delta m_F \in \{-1, 0, 1\}$. The allowed decays of an $F' = 3$ state are shown in Figure 1.3.

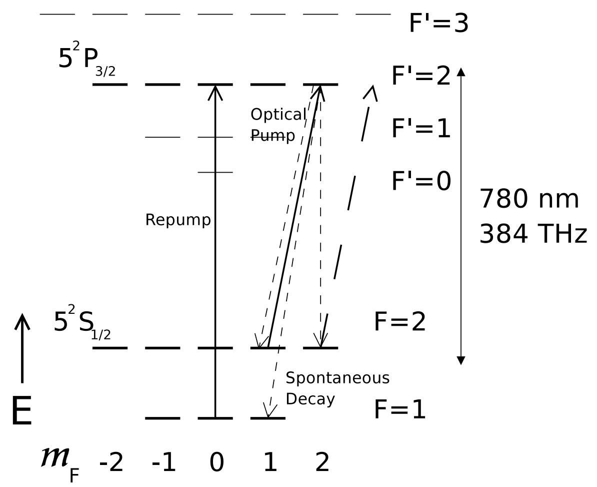


Figure 1.3: The repump transition, optical pumping transition, and spontaneous decay of the $F' = 2$ stretched state in $^{85}$Rb. Note that there is no optical pumping transition from the stretched $F = 2$ state.

While energy differences allow the selection of most transition, a specific $m_F$ transition, $\Delta m_F = -1, 0, 1$, can be selected by the polarization orientation, $\sigma+$, $\sigma-$ or $\pi$, and/or by splitting the energy degeneracy by the Zeeman effect. The Zeeman splitting for small magnetic fields is linear with the field and $m_F$ number. The linear constants are given in MHz/G in Figure 1.2.

A weaker magnetic dipole transition can be excited between the atomic hyperfine states with $\Delta n = \Delta l = 0$ As the magnetic transition dipole moment is on the order of $1/c$ weaker than the electric dipole moment, the Rabi oscillation and state transfer are much slower than for a similar optical transition.

Several transitions are used in our trapping apparatus as shown in Table 1.1. The optical pumping transition is used to prepare all the atoms in the upper hyperfine stretched state, $m_F$ maximized, in preparation for transfer to any other hyperfine ground state. This transition must be excited in combination with the repump light in order to populate the upper hyperfine states, similar to the repump requirement for the MOT pump transition.

| Name | Transition | Notes |
|---|---|---|
| | Rubidium-85 | |
| MOT Pump | $F = 3$ to $F' = 4$ | Red detuned |
| MOT Repump | $F = 2$ to $F' = 3$ | |
| Optical Pumping | $F = 3$ to $F' = 3$ | polarized for $\sigma+$ transitions |
| Absorption | $F = 3$ to $F' = 3$ | Used for imaging |
| RbSS | $F = 2$ to $F = 3$ | 3.0GHz |
| | Rubidium-87 | |
| MOT Pump | $F = 2$ to $F' = 3$ | Red detuned |
| MOT Repump | $F = 1$ to $F' = 2$ | |
| Optical Pumping | $F = 2$ to $F' = 2$ | polarized for $\sigma+$ transitions |
| Absorption | $F = 2$ to $F' = 2$ | Used for imaging |
| RbSS | $F = 1$ to $F = 2$ | 6.8GHz |
| | Lithium-6 | |
| MOT Pump | $F = 2$ to $F' = 3$ | Red detuned |
| MOT Repump | $F = 1$ to $F' = 2$ | |
| Absorption | $F = 2$ to $F' = 2$ | Used for imaging |

Table 1.1: Optical and microwave transitions in use.

## 1.4 Adiabatic Transitions

Rabi oscillations are sensitive to the timing and intensity of the pulse, making a complete transfer of population directly between stable states difficult. Adiabatic transfer is another method to exchange populations of atoms.[18] By slowly changing the frequency of the incident pulse, the atoms can be cajoled into a different spin state. This can be seen as keeping the atoms in the ground state of an avoided crossing between the combined system of atoms and photons as per Figure 1.4.

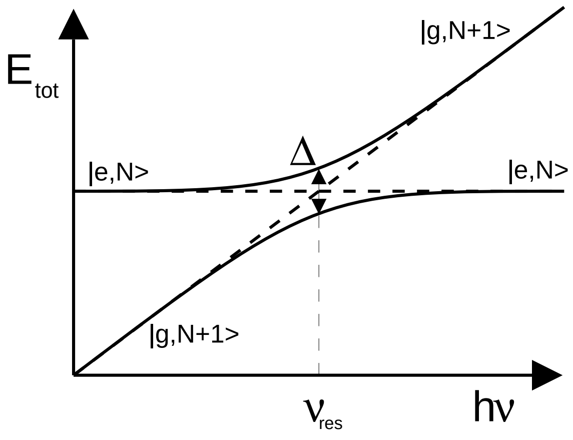


Figure 1.4: When considering the entire quantum system of radiation incident on an atom, the energy of the photons at a resonant frequency will create a degeneracy between the excited and ground states of the atomic transition. Dependant on the strength of the radiation field, an avoided crossing and energy gap, Δ, will form. If one sweeps the frequency of the incident photon up slowly across the resonance, by the adiabatic theorem the atom will stay in the ground state of the combined potential.[18] The rate at which one can sweep the frequency is inversely proportional to Δ.

The probability of state transfer using adiabatic transitions is given by,

$$P = 1 - e^{\frac{-(2\pi\Omega)^2}{\frac{df}{dt}}} \tag{1.2}$$

where $f$ is the frequency of the incident radiation and $\Omega$ is the rabi frequency at the transition. This probability of success assumes that the incoming radiation field ramps in energy from $-\infty$ to $\infty$. In practice, starting the ramp several linewidths away from the transition is sufficient. By fitting an exponential curve to a measurement of atomic state transfer, the Rabi frequency of the transition at resonance can be determined.

## 1.5 Feshbach Resonance

Dipole-dipole, electron spin, and nuclear spin interactions form weakly attractive potentials between alkali species. At ultracold temperatures, the weak interatomic potentials of alkali atoms allow the production of stable dimer molecules. Both homonuclear and heteronuclear dimers can be formed. Of $^{85}$Rb, $^{87}$Rb, and $^{6}$Li, only the potentials of $^{85}$Rb$^{6}$Li have not been measured.[3]

The interatomic potential of these atoms is well approximated by the Born-Oppenheimer potential.[2] This potential is similar, and approximates at long range, the van der Waals potential. The singlet and triplet potentials between two alkali species are differentiated by the electron spin interaction between the atoms. Antialigned spins in the singlet potential, $\vec{S}_1 + \vec{S}_2 = 0$, produces a much deeper potential than the triplet potential, $|\vec{S}_1 + \vec{S}_2| = 1$.[2] The factors of these potentials are difficult to calculate, requiring many approximations. This difficulty spurs empirical measurements of these potentials.

Ultracold atoms will slowly form dimer molecules spontaneously through inelastic three body collisions in the trap. Heating from these collisions can eject atoms from the trap. Another mechanism for molecule production is 3-body decay where one of the particles is a photon. In this process, two colliding atoms absorb a photon which promotes them into an electronically excited bound molecular state. These collisions will typically decay into very hot free atoms and leading to a loss of trapped atoms. For a MOT, light assisted collisional loss and heating are the dominant loss mechanism at high densities, and the light assisted collisional loss rate has been measured for mixtures between pairs of $^{6}$Li, $^{85}$Rb, and $^{87}$Rb.[4, 12]

To measure the energy levels of the atoms, and thus the interatomic potentials, resonances between two atomic states are found. Two methods have been used: Feshbach resonances, and photoassociation. Photoassociation allows resonances to be found between all bound molecular states.[10] Photoassociation however requires two widely tunable laser sources. Feshbach resonances are easier to find, but they cannot probe deeply bound states. Feshbach resonances couple the unbound state of an atom pair to the bound state of another pair with a different spin state as per Figure

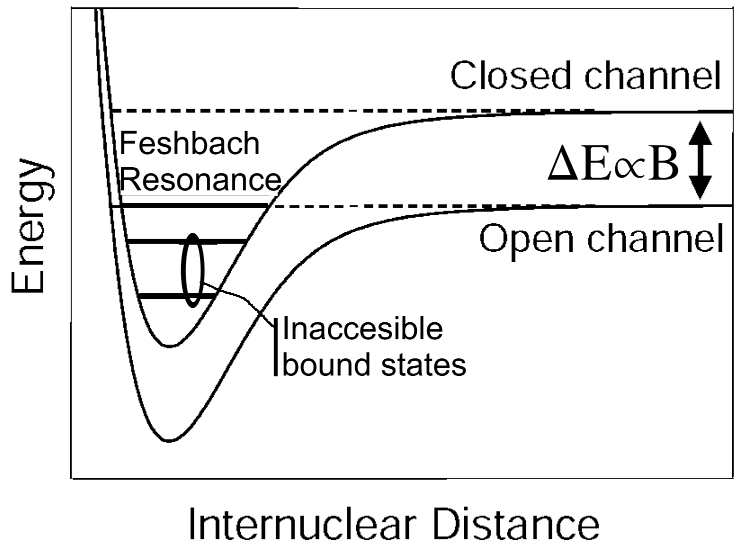


Figure 1.5: Feshbach resonances occur when the bound state of an interatomic potential is made degenerate with the unbound state of a shifted interatomic potential. The shift is produced by the Zeeman effect and is tunable by an external magnetic field. The strength of the magnetic field limits the number of states for which a Feshbach resonance can be found. Deeply bound states are inaccessible.[2]

1.5. An external magnetic field can tune to the resonance by exploiting the Zeeman splitting of the different spin states.

At resonance, atom loss due to three body collisions and the production of molecules is greatly enhanced. Due to this effect, an excess loss of atoms from the trap can be used to identify the resonances and their linewidths.

# Chapter 2

# Measurements

The steps required to take Feshbach resonance spectra are:

- Produce dual lithium and rubidium MOTs
- Trap lithum and rubidium in a dipole trap
- Calibrate the current of the Helmholtz coils to the magnetic field strength at the dipole trap
- Prepare rubidium into a single hyperfine state by optical pumping and adiabatic transfer
- Find known homonuclear Feshbach resonance structures
- Take Feshbach resonance data for LiRb

At the beginning of this project, the dual MOT setup was in place and rubidium was successfully trapped in a dipole trap.

## 2.1 Dipole Trapping and Loading

Details of dipole trap loading and measurements thereof can be found in Will Gunton's thesis[9]. In summary, early in 2008, an IPG brand 20W 1064nm fiber laser was installed to provide a dipole trapping beam. While this beam trapped rubidium, no lithium loading was measured in the trap at the time. An SPI brand 100W 1075nm fiber laser was a acquired to provide a second dipole trap in which to cool the lithium atoms prior to loading in the original 20W trap. This trap at power outputs of 30W was damaging the glass of the trapping cell. A crossed trap was devised to spread the heat load among two points on the cell, resulting in about quadruple the trap depth of the IPG trap. As the installation and commissioning of this trap has just completed, rubidium has been trapped and lithium trapping is being attempted.

To achieve an efficient transfer of atoms from a MOT to a dipole trap, there must be a large overlap between the traps, and the load timing must be optimized to allow for cooling of the MOT atoms while preventing trap losses. The trapping efficiency can be parameterized by the initial loading rate into the dipole trap and a loss from the dipole trap due to collisional heating from the MOT pump and repump light. The loading rate can be optimized by correctly positioning the MOT with respect to the dipole trap and by adjusting the detunings. The loss rate can be reduced by lowering the intensity of the MOT light around the center of the dipole trap.[11]

While it is unfeasible to change the geometry of 6 MOT pumping beams to provide a small minimum in the pump light, this is easily done with the repump. During normal MOT loading operations, a conventional repump beam is used. When loading the dipole trap, a shadow repump beam is to be used that images a shadow onto the dipole trap. Thus, the MOT continues to receive repump light while atoms in the dipole trap receive only pump light. As atoms in the dipole trap decay into the lower hyperfine ground state, they become dark to the MOT light, and light assisted collisional losses are greatly reduced. Using a dipole trap at 784nm, near detuned from the 780nm $^{85}$Rb $D_2$ line, Kuppens et al. found that a shadow repump beam doubled the loading into the dipole trap.[11] With our 1075nm dipole trap, off resonant repumping from the trapping beam is negligible as compared to a 784nm trap, and a shadow repump should have a greater effect.

## 2.2 Imaging

This experiment uses two methods for imaging atoms: fluorescence imaging and absorption imaging. Atoms trapped in a MOT are constantly absorbing and remitting light. This remitted fluorescent light is emitted isotropically and is proportional to the intensity of the MOT beams and the trapped atom number. Calibrated photodiodes and an Apogee Alta U32 CCD camera are used to measure the fluorescence and image it.[12]

Absorption imaging images the shadow left in a beam of resonant light by atoms. This technique can make use of short light pulses to image specific atom populations in a dipole trap. The short pulse ensures that the trap has not evacuated during image exposure. Absorption imaging is not as accurate as fluorescence imaging as a measure of atom number.

A Pixelink PL-A741 CMOS camera is used for absorption imaging. This camera is capable of taking consecutive frames rapidly, thus allowing the taking of a background image under nearly identical conditions as the signal image.

For measuring atom loss due to Feshbach resonances, atoms will be recaptured from the dipole trap into the MOT, and a fluorescence measurement will be taken. However, for the verification of state transferring and magnetic field calibration, imaging of specific atomic states through absorption will be required.

## 2.3 Magnetic Field Calibration

Feshbach resonance spectra are measured as a function of the applied magnetic field. The anti-Helmholtz field coils in this experiment can be reversed to apply a Helmholtz field that is approximately constant across the cell volume. A UBC Electronics Shop 25A coil driver is used to set the coil current. The inhomogeneity in the magnetic field across the volume of the dipole trap is negligible in this configuration.

As the exact geometry of the coils and the trap are unknown, the magnetic field at the dipole trap will need to be calibrated to the current setpoint of the coils. The Zeeman splitting of rubidium provides a magnetic field dependant probe at the dipole trap volume. Spectra of the hyperfine levels will be taken by pulsing microwave frequencies at the rubidium for different magnetic fields and excitation frequencies. The atoms will be prepared in the lower hyperfine state in the dipole trap. A microwave pulse will transfer the atoms into the upper hyperfine state where an absorption image is taken. To find the resonances, the microwave pulse is to last on the order of one second to allow decoherence of the Rabi oscillations and a broad spectral peak to form. Shorter or less powerful pulses on the order of $10^3$ less emitted energy, allow the measurement of individual Rabi oscillations and the central peak of the resonance. A fit to the decoherence of the Rabi oscillations can also be made to measure the linewidth of the resonance, and possible line broadening within the dipole trap. Several calibrations should be performed to correct against any long term drift in the coil driver or repositioning of the dipole trap in the Helmholtz field.

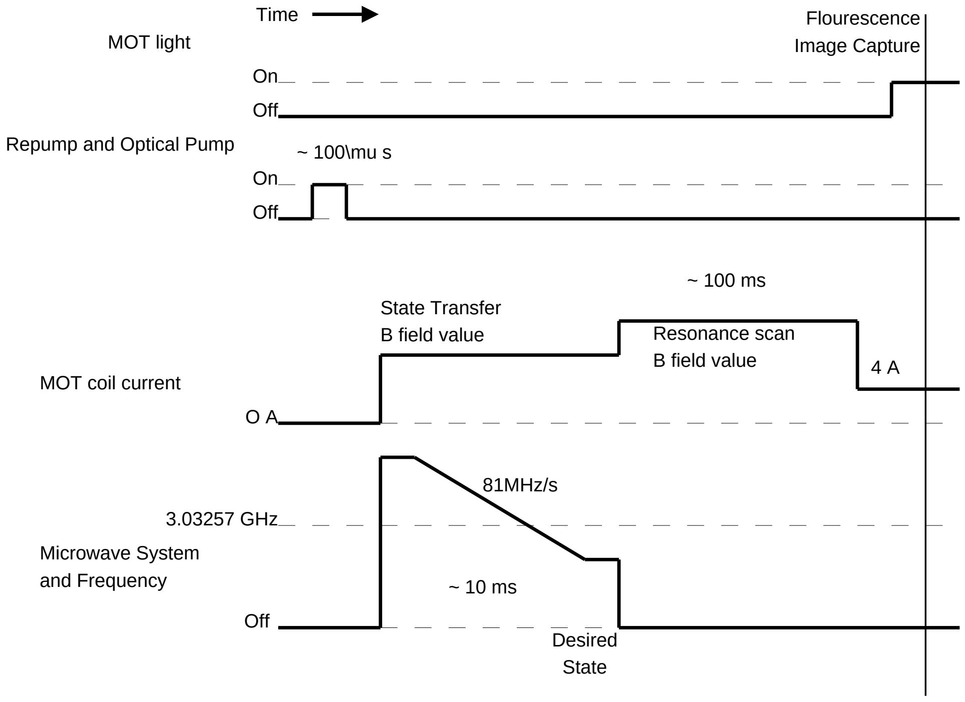
## 2.4 State Preparation and Feshbach Resonance Spectroscopy

Following the magnetic field calibration, algorithms for preparing rubidium into a specific hyperfine state are required. After loading into the dipole trap, the atoms are distributed in a mixture of $m_F$ states in the ground hyperfine state, $F = 2$ for $^{85}$Rb. Optical pumping and repump light is then shone onto the atoms to place them in the upper hyperfine stretched state. From there, adiabatic transfer under a magnetic field will take them to the desired spin state as shown in figure 2.1(b).

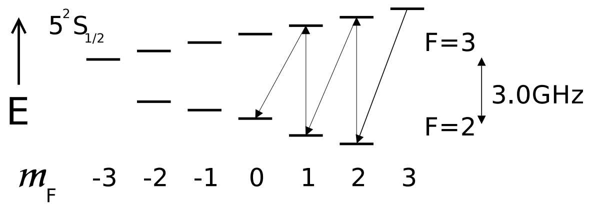
Once in the desired spin state, the magnetic coil currents are changed to explore a Feshbach resonance, then the atoms are recaptured in a MOT and imaged. Approximate timings for the procedure are shown in Figure 2.1(a). Only a short pulse is required to induce sufficient Rabi cycles to transfer atoms by optical pumping.

The output power of the RbSS microwave system is approximately $8.2 \pm 2.3$W/m$^2$ for $^{85}$Rb and $2.4 \pm 0.5$W/m$^2$ for $^{87}$Rb as per Table 11.2. This gives Rabi oscillations frequencies at the avoided crossing of $2.46 \pm 0.35$kHz and $1.98 \pm 0.20$kHz. To adiabatically transfer atoms with 99.5% efficiency, the microwave systems have to be swept at under 45MHz/s and 29MHz/s respectively as per Equation (1.2). Assuming linewidths on the order of the Rabi oscillation frequency, a magnetic field strength will be chosen far from a Feshbach resonance to separate each resonance by about 10 linewidths of the hyperfine resonance so that the sweep can be stopped between two lines. The adiabatic transfer process will then take on the order of 10-100ms.

Once atoms are transfered, the magnetic field is tuned to a possible Feshbach resonance, and atom loss is measured.



(a) Experimental procedure for measuring Feshbach resonances



(b) An example of the states traversed during adiabatic transfer.

Figure 2.1:

# Chapter 3

# Results and Progress

## 3.1 Optical Pumping Beam

A path for the optical pumping beam was installed. The beam takes light from the first Rb pump amplifier on the experiment table, downshifts it to meet the $F = 3$ to $F' = 3$ resonance, and circularly polarizes it down the vertical axis of the trapping cell, collinear with the Helmholtz magnetic field. A diagram of the setup is shown in Figure 3.1. The beam can also couple lithium pump light down the same path. As there is no lithium pump acouto-optical modulator on the experiment table, the lithium light is not frequency downshifted onto resonance.

As the optical pumping light is right-circularly polarized down the z-axis, the magnetic field within the coils must be oriented in the positive z direction to excite $\sigma+$ transitions.

## 3.2 Absorption Imaging

While lithium has not yet been trapped in the dipole trap, rubidium has been trapped for about one year. In the fall of 2008, several attempts at obtaining consistent absorption image data were attempted with mixed success. While state transfer using the microwave system was observed, imaging results were inconsistent. In the time between a signal image and a background image, a diffraction pattern on the Pixelink camera image would shift, preventing any usable measure of atom transfer. A sample background subtracted image is shown in Figure 3.2.

Simple averaging over multiple image sets was not a satisfactory solution. In combination with other equipment around the MOT cell, Bruce Klappauf moved the absorption image setup over the winter. In its new location, the camera and optical elements are more securely fastened resulting in less vibration. Additionally, Bruce rewrote the absorption imaging software to fit the peak of atoms in the image instead of summing the entire image. Since this change, the Pixelink camera has been providing adequately consistent signal in its images.

In it's new position, the absorption imaging beam is nearly collinear with the arms of the SPI 100W laser dipole trap, and perpendicular to the IPG 20W laser dipole trap. Thus for the same number of trapped atoms, the absorption is more intense for the enfilade atoms in the SPI trap as shown in Figure 3.3. Today, adequate signal is being gained from the absorption image setup for calibration purposes.

## 3.3 Magnetic Field Calibration

In autumn 2008, evidence of state transfer by the microwave system was observed. However, in attempting to find the value of the $^{85}$Rb hyperfine resonance at no magnetic field, problems with the absorption imaging system were encountered. Figure 3.5(a) shows the best data run taken during this time. By pulsing the microwave system for 1s at frequency intervals of 200kHz, the resonance was found as expected at 3.0357GHz.

Following the difficulties with obtaining an absorption image in the fall, attempts at calibrating the Helmholtz magnetic field were delayed until late February 2009. Attempts to transfer atoms from the lower hyperfine state to the upper hyperfine state failed for several weeks. Although some software glitches were found, they were fixed. Neither transfer by adiabatic or rabi oscillation methods worked: see Figure 3.5. Other than the new absorption imaging setup, software, and SPI dipole trap, conditions the experimental sequence were verified to be identical between both measurements.

In addition the failure to get a positive signal of state transfer, atoms were intermittently observed in the upper hyperfine state without deliberate state transference. This can be seen in the images of Figure 3.3. This transference was observed with the microwave system entirely switched off, and is intermittent. No known cause has yet been found.

While I have been writing this thesis, Benjamin Deh has successfully measured state transference using the microwave system by adiabatic transfer. However, the intermittent problem of finding unwanted atoms in the upper hyperfine state persists.

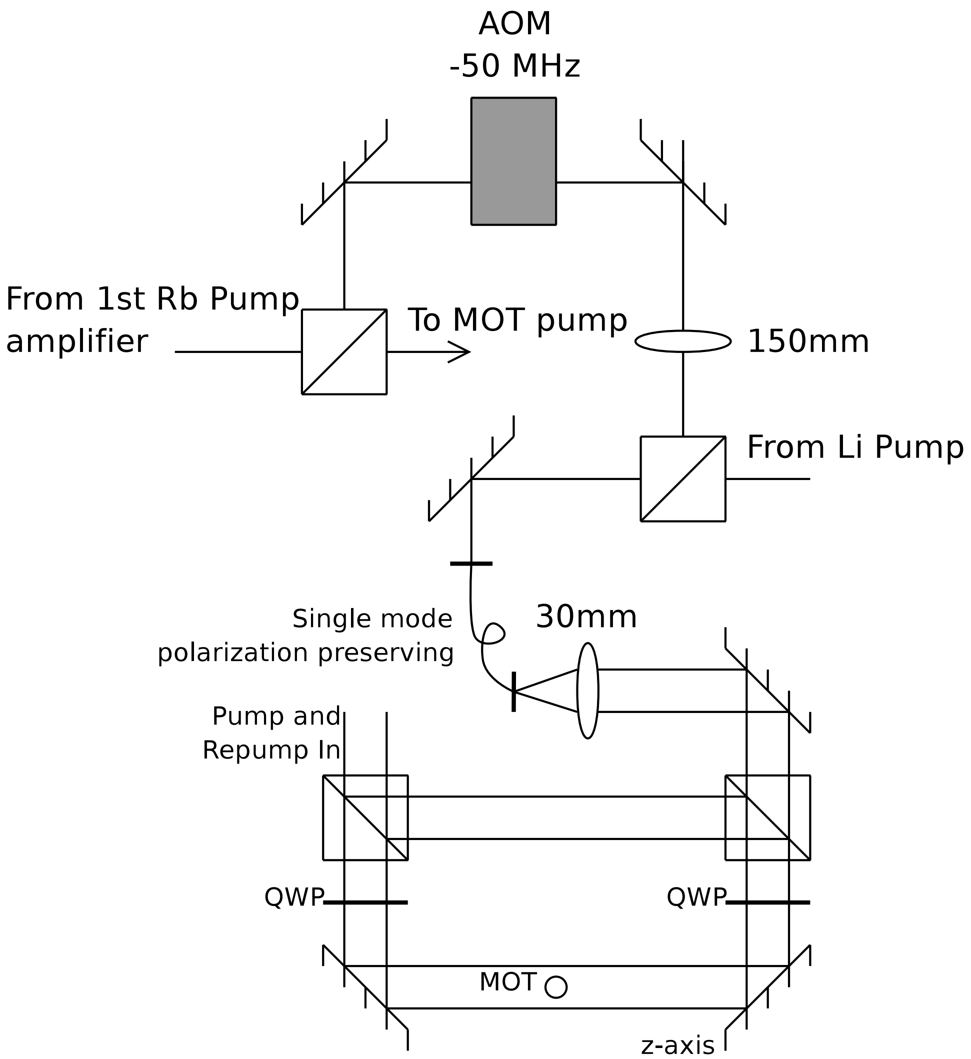


Figure 3.1: Path of the optical pumping beam. The beam couples resonant rubidium light, or lithium MOT pump light circularly polarized down the magnetic field axis of the trapping cell.

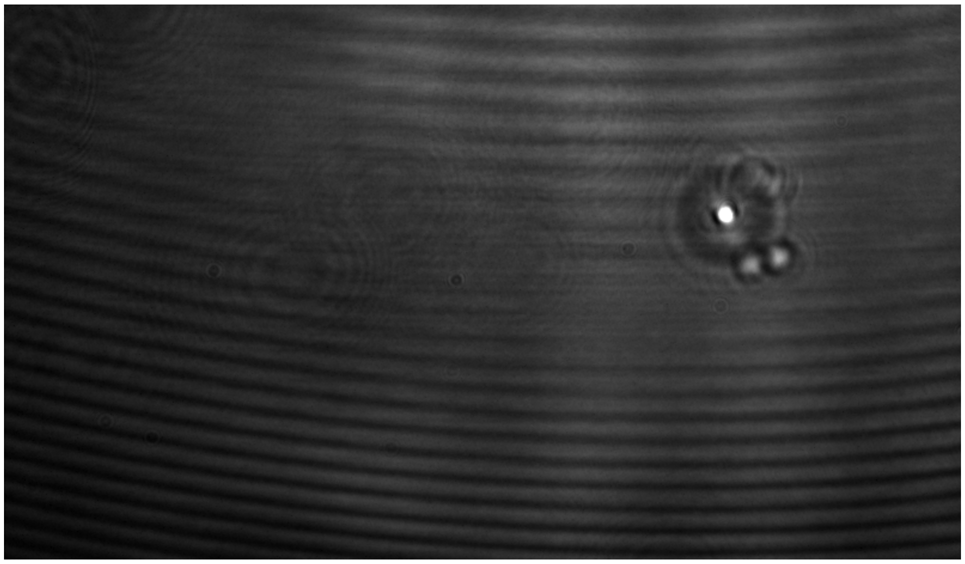
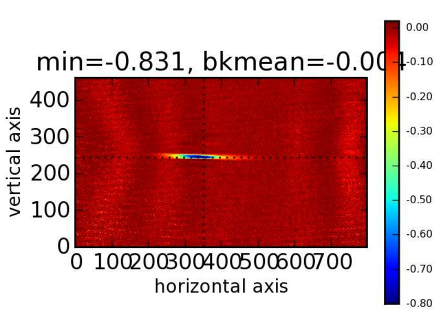
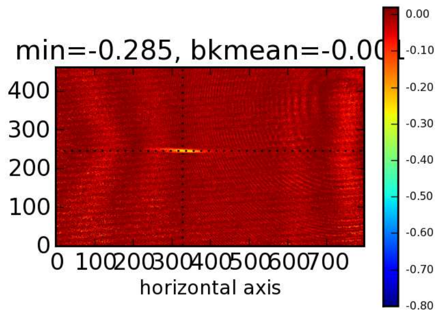


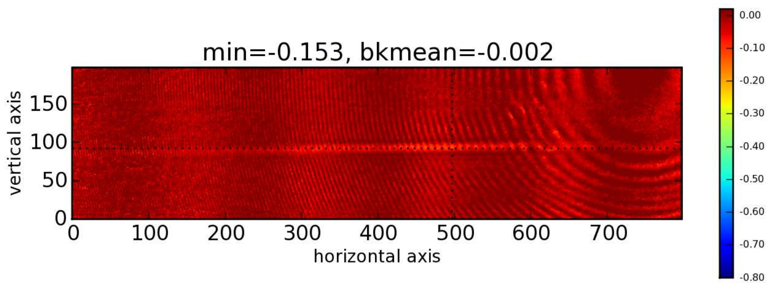
Figure 3.2: A raw absorbtion image.



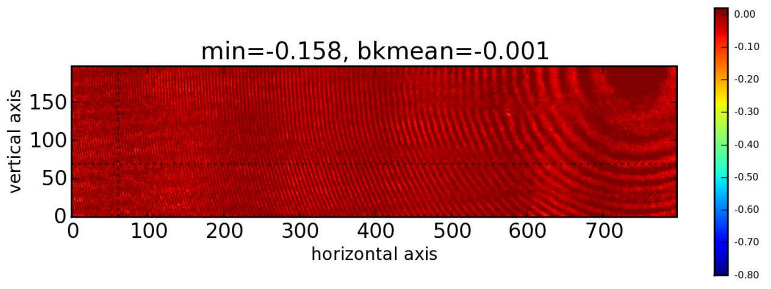
(a) Absorption of SPI 100W laser dipole trap with $^{85}$Rb atoms in the upper hyperfine state.



(b) Absorption of SPI 100W laser dipole trap with $^{85}$Rb atoms prepared in the lower hyperfine state.



(c) Absorption of IPG 20W laser dipole trap with $^{85}$Rb atoms in the upper hyperfine state.



(d) Absorption of IPG 20W laser dipole trap with $^{85}$Rb atoms prepared in the lower hyperfine state.

Figure 3.3: Background subtracted absorbtion images.

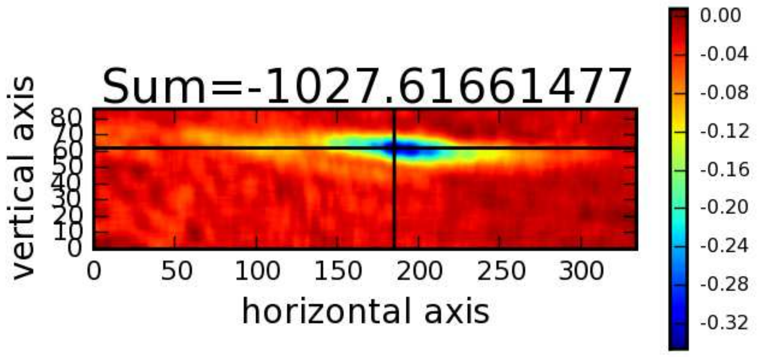
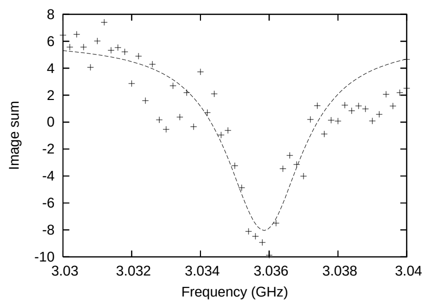
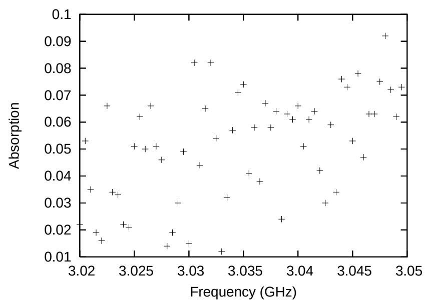


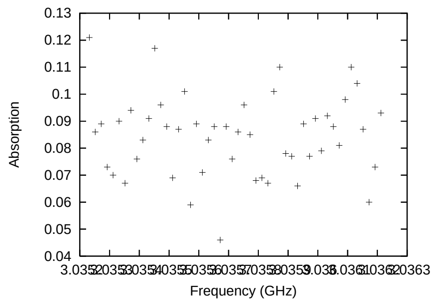
Figure 3.4: Absorption of IPG 20W laser dipole trap with $^{85}$Rb atoms in the upper hyperfine state in October 2008.



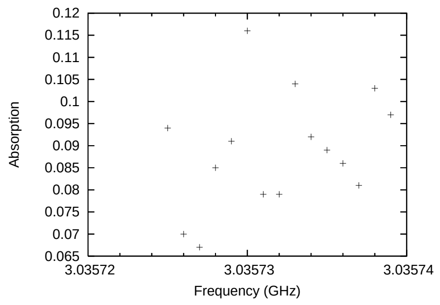
(a) Microwave system state transfer, October 2008.



(b) Search for $^{85}$Rb hyperfine state transfer. 500kHz intervals. March 2009.



(c) Search for $^{85}$Rb hyperfine state transfer. 20kHz intervals. March 2009.



(d) Search for $^{85}$Rb hyperfine state transfer. 1kHz intervals. March 2009.

Figure 3.5: Attempts made to find hyperfine transition resonances using the microwave system in March 2009 failed where success was had the previous fall. Using identical 1s microwave pulses, a clear resonance is seen in the first image, but no resonance of any width is seen in the next three. Attempts to transfer atoms adiabatically between the hyperfine states also failed. Note that in the latter 3 images, resonance curves are expected to increase absorption up the y-axis.

# Part II

# Towards Trapping and Photoassociation Spectroscopy

# Chapter 4

# Motivation

While the Feshbach resonance experiment has been fraught with delays, I have aided with setting up apparatus for efficient dipole trapping and photoassociation experiments. A shadow repump beam was installed to improve dipole trap loading as discussed in Section 2.1. Hardware for the Apogee camera was upgraded to allow fast image capture of successive frames. Finally, software was developed to interact with the hardware required for the single photon photoassociation experiment using the same trap as the Feshbach resonance experiment.

# Chapter 5

# Shadow Repump

A lens apparatus was prepared for illuminating the trapping cell with a shadow repump beam. A diagram of the apparatus is shown in Figure 5.1. The lens tube couples repump light from a fiber and collimates it while imaging a sewing needle onto the plane of the MOT and dipole trap. An iris at the imaging plane ensures that the collimated beam can be narrowed sufficiently to clear obstructions near the MOT cell and to prevent repump light from being scattered into the shadow region.

Taking images with the Apogee U32 camera, the focus of the shadow and the collimation of the beam were adjusted. Intensities for various exposure times from 10ms to 1s were taken for the lightest region and darkest region of the shadow image plane. Fitting the slope of the intensity versus exposure time for three data runs, the bright region is a factor of $220 \pm 20$ times brighter than the dark region as shown in Figure 5.2.

The shadow repump lens tube has two collars to adjust the focus, collimation, and shadow orientation of the beam. The imaged needle is located between the two collars. After adjusting the collar farthest from the fiber input for beam collimation, beam focus must be adjusted by rotating both collars simultaneously to adjust the position and orientation of the needle. The needle is to be aligned normal to the glass cell so that the shadow overlaps with itself in any reflections.

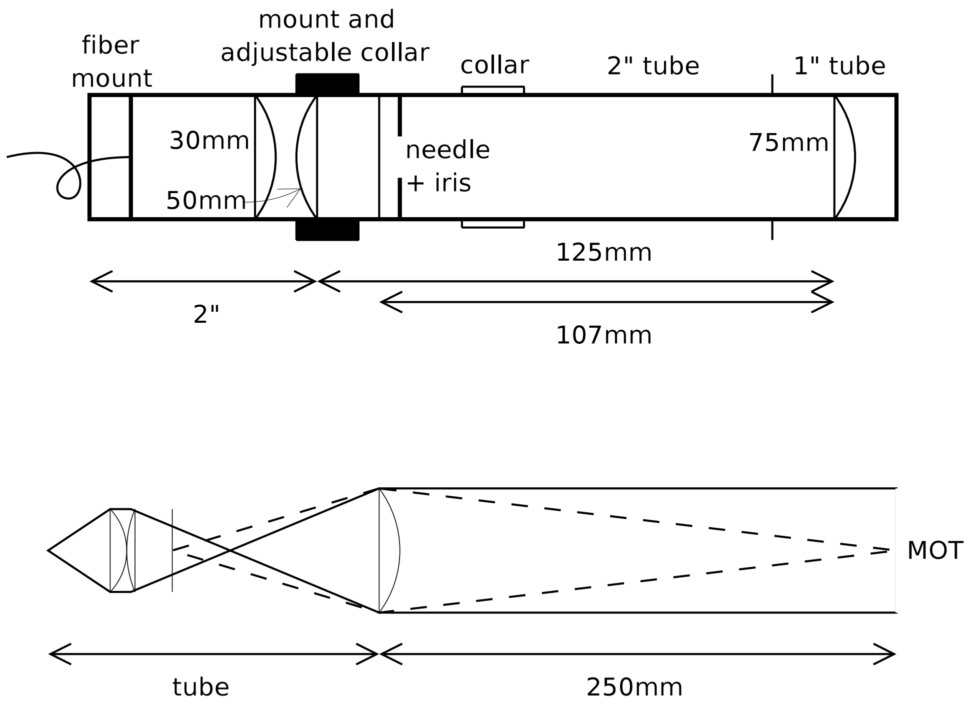


Figure 5.1: A diagram of the shadow repump lens tube above, and a ray diagram of the shadow focusing below.

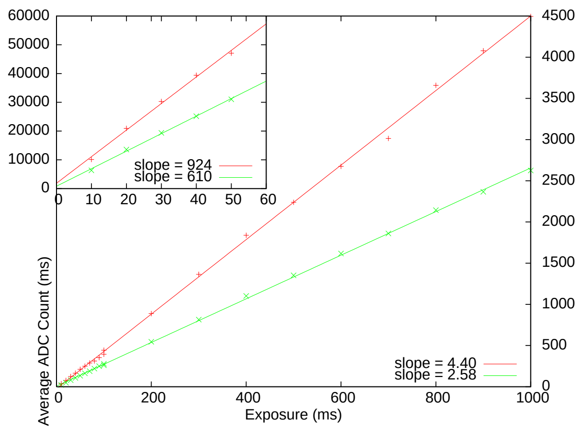


Figure 5.2: Data for the depth of the Shadow Repump shadow. The large graph shows the average brightness of a 15×15 pixel area of the shadow image while the upper left graph shows the same data for the bright area for two data sets. Comparing the slopes of the two graphs, the shadow is 210 or 236 times darker than the bright area of the beam.

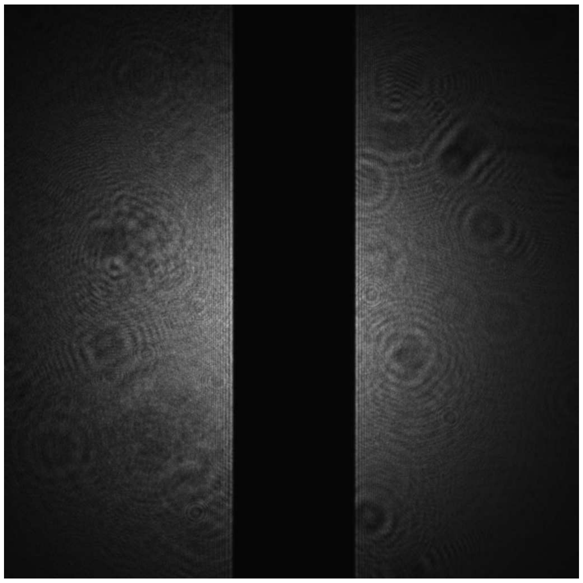


Figure 5.3: The image plane of the shadow repump beam, 40ms exposure.

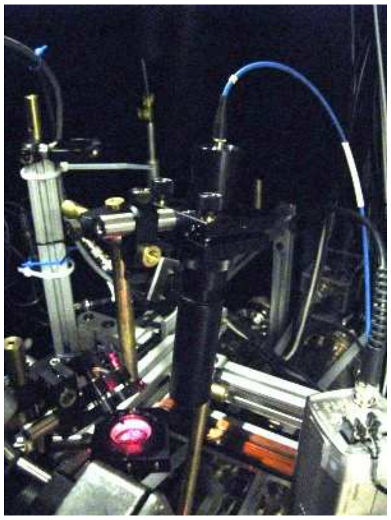


Figure 5.4: An image of the installed lens tube.

# Chapter 6

# Apogee Camera

The Apogee Alta series of cameras allow multiple images to be captured on their CCD without requiring readout in between. This is called kinetics mode. Between exposures, the charge on the CCD is transfered to an unexposed region of the CCD. Once the CCD is fully exposed, the image is readout, and the image can be post-processed to separate the individual exposures. This limits the camera speed to the shutter speed instead of the readout and digitization speed.

Raymond Gao had attempted to make the Kinetics mode work as described in his thesis.[6] The experiment's requirements were to externally trigger each shutter opening prior to readout, and to preset the duration of each exposure using software control. This requires hardware triggering of both pin 1 and pin 5 of the Apogee camera I/O port with the `TDIKineticsEach` variable set to `True` and `TDIKineticsGroup` set to `False` when initializing the camera.[1] A trigger is sent to pin 1 of the port for each opening of the shutter, and a trigger is sent to pin 5 to initiate readout of the CCD. A second isolator circuit had to be installed identical to that and in the same box as Ray Gao had installed for pin 1. This circuit is designed to reduce the input voltage from a 5V TTL to a 3.3V LVTTL signal, to reduce the effect of line noise on camera triggering, and to mate BNC cable from the UTBus software controlled outputs to the 8-pin DIN connector of the Apogee I/O port. A diagram of the circuit is shown in Figure 6.1.

While reworking the Apogee isolator circuit, I had failed to make a grounding connection in the box's power supply, thus floating both pin 1 and pin 5 at 8.3V for a period of a few minutes.

While testing the camera in kinetics mode, each shutter operation would trigger properly using the pin 1 trigger, but the readout would not be triggered by pin 5. The signal into of pin 5 was verified to go high on trigger, but the camera would not respond. The camera status was verified using its `ImagingStatus` codes. Prior to the first trigger, the camera would reply with code 5, `Apn_Status_WaitingOnTrigger`. While triggering the individual exposures, the camera would toggle between status codes 1 `Apn_Status_Exposing` and 2 `Apn_Status_ImagingActive` as expected. Once

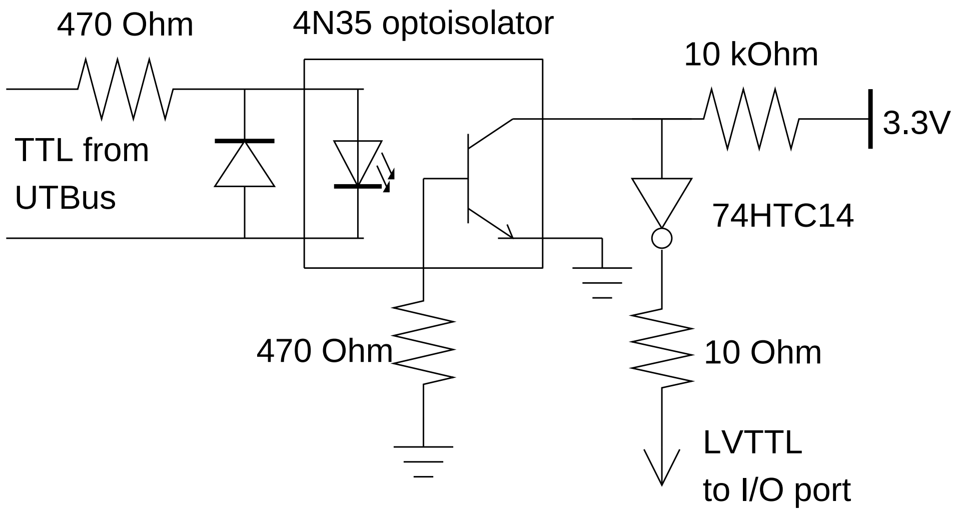


Figure 6.1: A protection circuit for the Apogee Camera I/O pins.[6]

the preset number of exposures has been triggers, pin 1 no longer has an effect, and the camera waits for a trigger from pin 5 to begin readout. However, the cameras status remains unchanged when triggering. A hardware problem with pin 5 is suspected, possibly due to the overvoltage condition imposed on it.

# Chapter 7

# Photoassociation Software

The single photon photoassociation experiment is designed to probe the energy spectrum of unstable first excited states of LiRb using a carefully tuned laser. This laser is locked to an ultrastable femtosecond frequency comb. Two frequencies characterize the comb as shown in Figure 7.1: the offset frequency $f_o$, and the repetition rate $f_rep$.

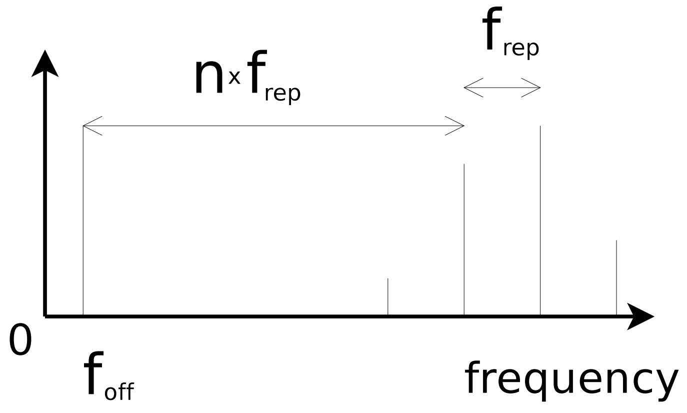


Figure 7.1: Schematic of a frequency comb. $f_o = f_{line} \% f_{rep}$ where $f_{line}$ is the frequency of any spectral line.

To tune the photoassociation laser, the repetition rate of the frequency comb need to be modulated and the locks verified. The rep rate is controlled by an HP8663A function synthesizer locked to the lab's 10MHz rubidium atom clock reference. The comb's lock to the rep rate and to the offset frequency are verified by Agilent 53132A frequency counters. The synthesizer and counters can be software controlled by a GPIB connection. The narrow band photoassociation laser is locked to the comb at a constant offset of $f_o$ from one of the comb lines. To check which comb line the laser is locked to, a Bristol Series 721 wavemeter is used. The wavemeter uses a USB 2.0 interface.

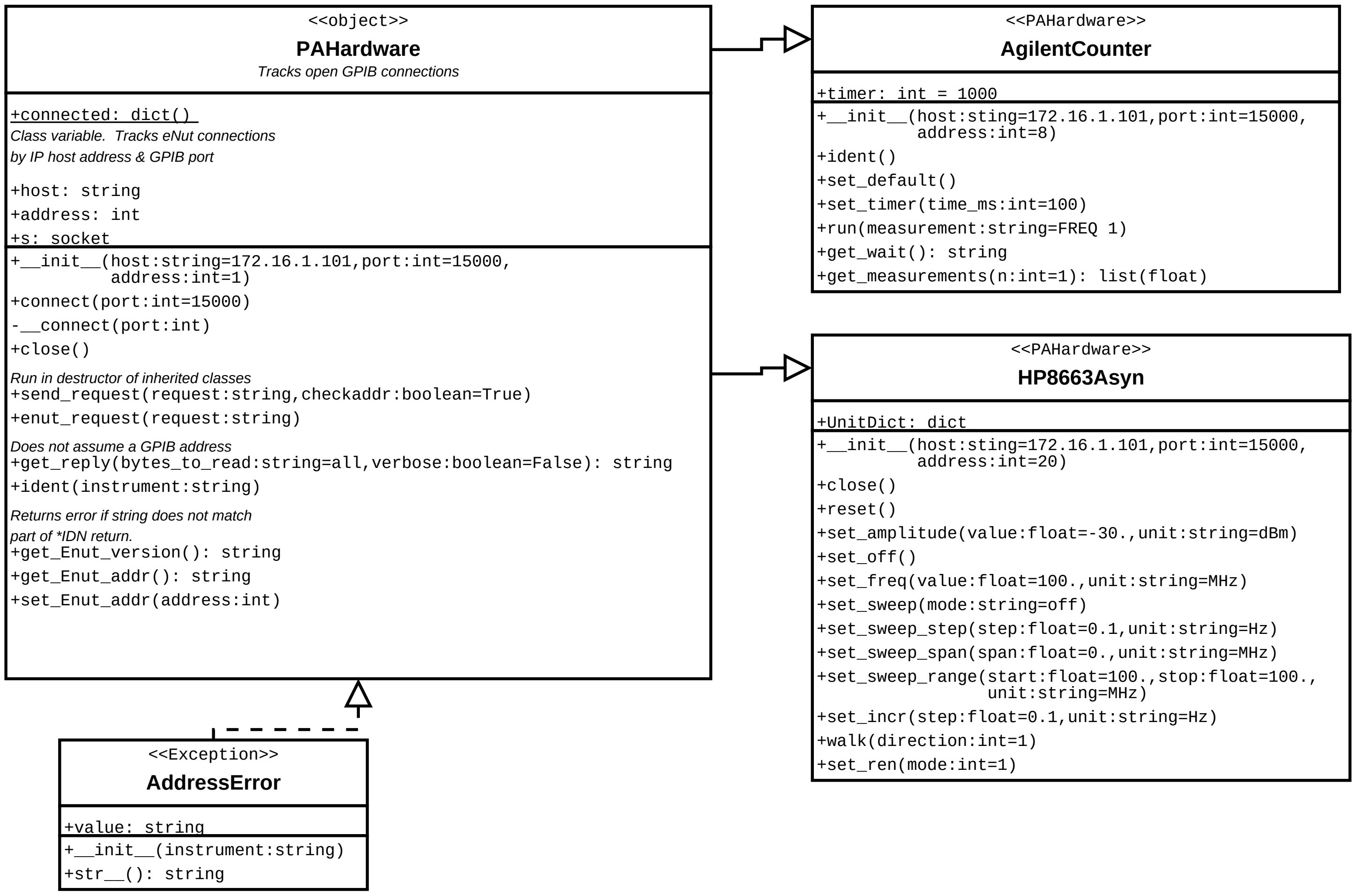

<<object>>
PAHardware
Tracks open GPIB connections
+connected: dict()
Class variable. Tracks eNut connections
by IP host address & GPIB port
+host: string
+address: int
+s: socket
+__init__(host:string=172.16.1.101,port:int=15000,
address:int=1)
+connect(port:int=15000)
-__connect(port:int)
+close()
Run in destructor of inherited classes
+send_request(request:string,checkaddr:boolean=True)
+enut_request(request:string)
Does not assume a GPIB address
+get_reply(bytes_to_read:string=all,verbose:boolean=False): string
+ident(instrument:string)
Returns error if string does not match
part of *IDN return.
+get_Enut_version(): string
+get_Enut_addr(): string
+set_Enut_addr(address:int)
<<PAHardware>>
AgilentCounter
+timer: int = 1000
+__init__(host:sting=172.16.1.101,port:int=15000,
address:int=8)
+ident()
+set_default()
+set_timer(time_ms:int=100)
+run(measurement:string=FREQ 1)
+get_wait(): string
+get_measurements(n:int=1): list(float)
<<PAHardware>>
HP8663Asyn
+UnitDict: dict
+__init__(host:sting=172.16.1.101,port:int=15000,
address:int=20)
+close()
+reset()
+set_amplitude(value:float=-30.,unit:string=dBm)
+set_off()
+set_freq(value:float=100.,unit:string=MHz)
+set_sweep(mode:string=off)
+set_sweep_step(step:float=0.1,unit:string=Hz)
+set_sweep_span(span:float=0.,unit:string=MHz)
+set_sweep_range(start:float=100.,stop:float=100.,
unit:string=MHz)
+set_incr(step:float=0.1,unit:string=Hz)
+walk(direction:int=1)
+set_ren(mode:int=1)
<<Exception>>
AddressError
+value: string
+__init__(instrument:string)
+str__(): string

Figure 7.2: A diagram of the classes used to interface to GPIB hardware for the photoassociation experiments.

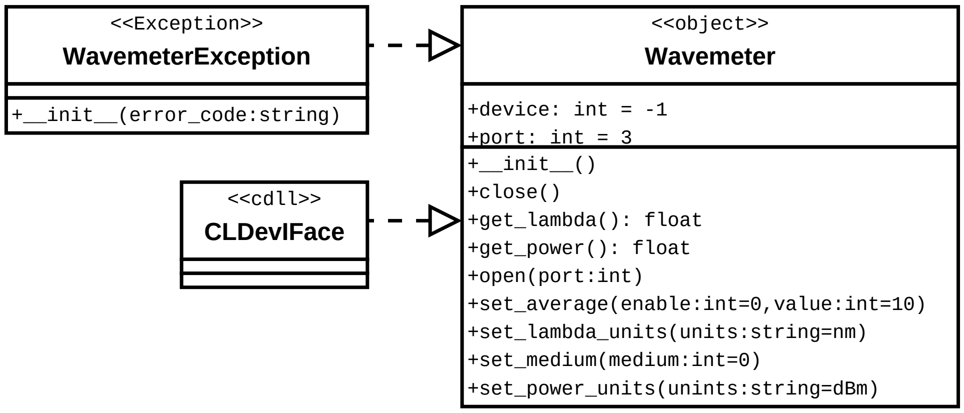


Figure 7.3: A diagram of the classes used to interface to the Bristol wavemeter the photoassociation experiments.

## 7.1 GPIB Control

The IEEE-488 standard, or GPIB bus standard, defines a simple addressable parallel interface between experimental devices. The Quantum Degenerate Gases lab uses a GPIB to ethernet interface called EtherNut which implements a simple socket interface to connect to GPIB devices remotely. To connect to a GPIB device, an EtherNut device is addressed and connected to over ethernet, the EtherNut hub is told to address a specific GPIB device, and commands can then be passed through the hub directly to the device.

As only one GPIB device can be addressed at a time on any GPIB bus, a Python class was developed to manage all open GPIB connections required for the photoassociation hardware interface. This class is shown in Appendix A.1. While the class is called `PAHardware`, it can handle any general GPIB connection among any number of EtherNut hubs. The class uses a `dict` structure to keep track of all open EtherNut connections and their respective GPIB addresses. Inherited classes do not need to keep track of any connections or to verify the GPIB address connected as this is handled by `PAHardware`. The implements all commands related to the EtherNut connection.

Of note to developers, the `ident()` (identify) method is very important to use within methods sending commands that may affect other GPIB instruments. The method matches a string argument to the instuments response of the `*IDN` GPIB command, and raises an exception if the string does not match. Any command defined by IEEE-488-2 of the form `*XXX` is shared between most GPIB instruments.

The reset command, `*RST`, was inadvertently sent to the ILX laser diode controller during debugging. The controller was set to factory defaults requiring reseting of the temperature and current control settings on many laser diode amplifiers. A missing line of code allowed the `*RST` to be sent to the default GPIB address, address 1, on the bus instead of address 8 connected to the frequency counter. The ILX controller was set to address 1. Now, the `AgilentCounter.set_default()` command implements `ident()` to avoid these problems.

## 7.2 HP8663A Function Synthesizer

The HP8663A function synthesizer can be controlled by GPIB command as if controlled from the front panel buttons. The synthesizer does not reply to any GPIB commands and is hardware set to GPIB address 20. Therefore the `ident()` command is useless for this instrument. In addition to setting single frequency and amplitudes, the frequency of the synthesizer can be stepped in two ways. First, a built in ramp function allows the synthesizer to step frequencies at a given step size and a selected rate. The Python software implements a 1ms step period where the stepped frequency can be set by the user with a precision of 0.1Hz. This ramping method is simplest to use to get to a new frequency quickly or to ramp the frequency up and down.

The preferred ramping method for the photoassociation experiment is to trigger each step in frequency. This is equivalent to pressing the increment buttons, or turning the selection knob, on the front panel of the synthesizer. An increment step if first selected, then the Python code used to control the instrument is used to ramp and control the frequency.

Code for the HP8663 can be found in Appendix A.2. It inherits from the `PAHardware` class.

## 7.3 Agilent 53132A Frequency Counter

Two Agilent 53132A Frequency Counters are connected to the same EtherNut hub as the synthesizer. These counters implement standard IEEE-488-2 commands and have programable GPIB addresses. Similarly to the frequency synthesizer, control of these instruments is similar to using the front panel controls. Although there is a `MEAS:` command expressly designed for software controlled measurements, this command does not allow control of gate timing. Every frequency measurement taken by the counter is written to an internal buffer that can be queried by the GPIB bus using the `DATA?` command. The buffer is updated only after a measurement is taken, so multiple `DATA?` queries may give the value of a single measurement.

The Python class `AgilentCounter` can set the gate time, units, and measurement source of the counters, and set them collecting measurements continuously. The `getMeasurement()` function returns a list of `n` float values timed to read consecutive counter measurements. To ensure that a measurement is available for reading, the Python code should wait at least `get_wait()` seconds after any `run()` command.

## 7.4 Bristol 721 Series Wavemeter

The Bristol 721 Series wavemeter has no front panel. Its measurement interface is a proprietary program or a C++ driver class connecting to the instrument over a USB 2.0 interface. As the wavemeter is located more than 5m away from the experiment's computer, a USB hub was acquired to allow a total cable length of 10m connect to the wavemeter. The `Wavemeter` class, in Appendix A.4, was modelled on the TekScope code written by Ovidiu Toader. Ovi's code using Python metaclasses and self-assembling code was greatly simplified in this implementation. The interface allows the setting of units, and the reading of input power and wavelength from the wavemeter.

In the `Wavemeter` class, every command is preceded by a connection reset command. Without this feature, the original implementation worked well from a laptop next to the wavemeter, even when using the full 10m of cable. However, the connection would irreversibly freeze when connected to the experiment's computer. The proprietary software Bristol provided freezes as well after receiving a handful of data packets. The reset command is a software workaround for the apparently bad USB bus on the experiment's computer. Attempts at shielding the USB cable from possible electronic noise in the vicinity failed to improve the situation. A hardware problem is suspected within the MOL experiment computer.

# Part III

# The Rubidium State Selector (RbSS) Microwave System

# Chapter 8

# Purpose

A microwave system has been constructed to promote the states of both $^{87}$Rb and $^{85}$Rb for the QDG Lab's dual species Magneto-Optical Trap (MOT) experiment. Promotion of the $^{87}$Rb hyperfine states requires a 6.83 GHz excitation while $^{85}$Rb requires 3.03 GHz. The system needs to be able to sweep across frequencies for adiabatic transfer, and to have stabilized power and integrated power outputs for calibration and microwave spectroscopy.

A second partially completed system was constructed to implement forced evaporative cooling of a future rubidium magnetic trap. This system is designed to simultaneously output 3.03GHz and 6.83 GHz whereas the first system is only capable of outputting a single frequency. Alternative component values for this system will be given where applicable.

The Rubidium state selector combines a phase lock loop, a power stabilization loop, a series of doublers and amplifiers, and an antenna system to deliver power to the MOT cell.

# Chapter 9

# Phase Locking

A highly stable and accurate microwave source was required. A rubidium atom clock at 10MHz clock is used by Direct Digital Synthesizers (DDSs), to provide a reference signal to lock a MiniCircuits ZX95-1200W Voltage Controlled Oscillator (VCO). The DDSs can produce both a highly stable single frequency and frequency ramps up to 100MHz. A phase lock loop (PLL) was designed to approach the required 3.0 and 6.8 GHz bands. The phase detectors available are UBC electronics shop E06-012-1s based around the AD9901[1] phase-frequency discriminator with frequency division by UPB1509PV and MC100E016FN counters. The counters operate up to 1GHz. Thus the PLL is designed to multiply the DDS frequency by 10 with further doubling steps to the required frequency.

The loop filters for the PLL was constructed from University of Texas PID controller boards.[14] Component values are shown in 9.1. The loop filters were designed to form a Type 3 third order PLL. This allows for zero phase error when ramping the set-point frequency of the loop.[7] The loop parameters were optimized to maximize the gain of the loop while keeping transient responses damped. An purely integrating loop filter would be unstable, therefore a proportional response component is used to dampen the filter. The open-loop transfer function for the PLL is given by

$$F(s) \approx \left(\frac{\tau_2}{\tau_1} + \frac{1}{s\tau_1}\right)\left(\frac{\tau_4}{\tau_3} + \frac{1}{s\tau_3}\right) \tag{9.1}$$

$$= \frac{s^2\tau_2\tau_4 + (\tau_2+\tau_4)s + 1}{s^2\tau_1\tau_3} \tag{9.2}$$

$$G(s) = \frac{K_d K_o F(s)}{s} \tag{9.3}$$

$$\tag{9.4}$$

[1] For future PLLs, UPB1507GVs can be substituted to place the operating frequency of the phase detector from 1GHz to 3GHz, alleviating the need for additional doublers.

| Component | Value |
|---|---|
| C2 | .01$\mu$F |
| C9, R2, R21, R37, R39<br>R50, R51, R57, R61 | 0 $\Omega$ |
| R40 | 49.9$\Omega$ |
| R14, R15 | 1k$\Omega$ |
| R3 | 3.83k$\Omega$ |
| R1 | 5k$\Omega$ pot |
| R10 | 49.9k$\Omega$ |
| R16 | 100k$\Omega$ |
| U1 | AMP03G |
| U2, U5, U6 | LF356 or OP37G |
| J2, J5 | BNC jacks |
| J4, pwr in | Molex 70543 series |
| Jumpered across U5 | |
| R52 | 49.9k$\Omega$ |
| R53 | 0.01$\mu$F |
| Decoupling Caps | |
| C12, C13, C20, C21 | 0.1$\mu$F |
| C10, C11 | 0.33$\mu$F |
| C41, C42 | 470$\mu$F |
| Voltage Regulation (opt.) | |
| U12 | LM78L |
| U13 | LM79L |
| C36, C37, C38, C39 | 10$\mu$F tant. |

Table 9.1: Component values for the RbSS Phase Lock Loops.

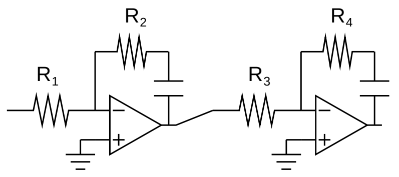


Figure 9.1: The core elements of a Type 3 PLL loop filter.

where $\tau_x$ is the time constant of $R_xC$ as in Figure 9.1. The filter was first optimized as a second order loop before adding the third integrator. The respone of this integrator was maximized, $\tau_1\tau_3$ minimized, within the regime that the loop still approximated the response of the original second order loop: $\tau_2 \ll \tau_4 \approx \tau_3$, giving a loop response of:

$$F(s) \approx \frac{s^2\tau_2\tau_4 + \tau_4 s + 1}{s^2\tau_1\tau_3} \tag{9.5}$$

$$\approx \frac{\tau_2}{\tau_1} + \frac{1}{s\tau_1} + \frac{1}{s^2\tau_1\tau_3} \tag{9.6}$$

where the first two terms are the original 2nd order loop response.

In a second order loop, the loop can be parameterized in terms of a damping constant, $\gamma$, and a loop gain, $K$.[7] Slightly overcritical damping with $\gamma = \frac{1}{\sqrt{2}}$ is preferred to both quickly dampen transient responses, and to reduce high frequency oscillations. The main limit to the gain is the effect of loop stability around the gain crossover frequency. As in any loop there is a propagation delay, there is a minimum frequency at which a $\frac{\pi}{2}$ phase shift will occur resulting in positive feedback of that frequency. For the loop to be stable, the feedback gain at that minimum frequency must be less than 0dB. As the gain at this frequency approaches 0dB, transient damping is impeded as it is at low gain. There is therefore a gain, for a given proportional versus integral path ratio, for which the damping of transients is maximized. Our loop filter was optimized to find the maximum gain for which the damping has $\gamma = \frac{1}{\sqrt{2}}$ by varying the gain and proportional versus integrator ratio as shown in Figures 9.3 and 9.4. The response specifications for this loop are shown in 9.2.

There are three controls to the PLL. First is the Servo Gain control on the Phase Detector. This directly controls the open loop gain of the PLL. Increasing the gain will cause loop instability at the gain crossover point. The gain should be set to approximately one third of that which will cause

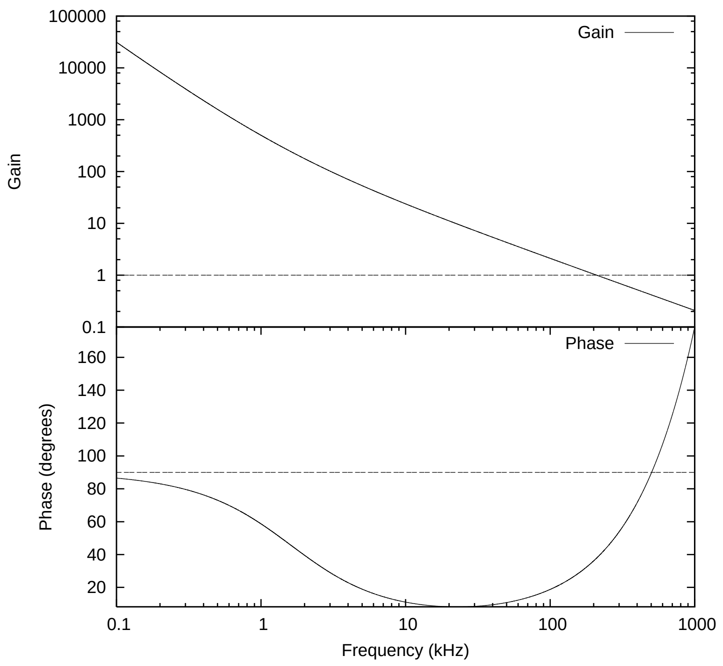
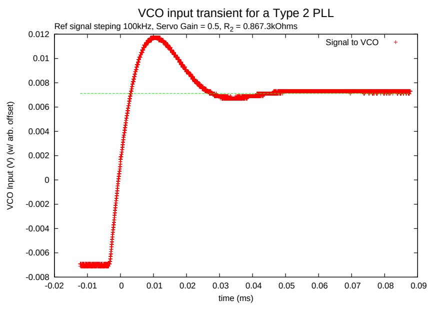
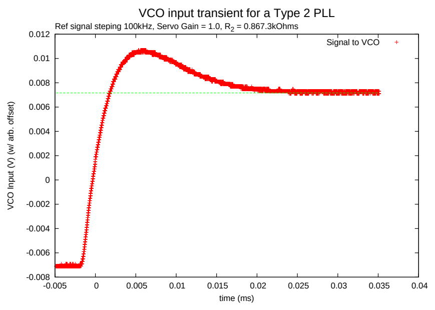


Figure 9.2: Bode plot of the RbSS phase lock loop. The curves are fit to the transfer function data of the RbSS system during optimization.

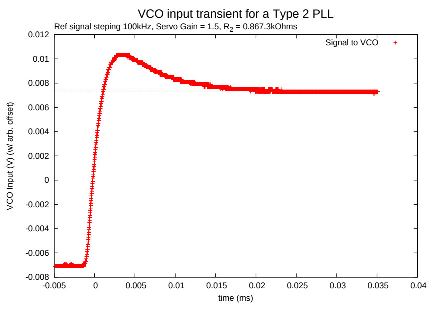
instability. Second, the DDS controls the setpoint frequency of the PLL. Thirdly, the frequency dividers inside the Phase Detector are controlled by the $N$ selector and the predividers. The $N$ selector divides the frequency by $2 \times N$ and the predividers independently divide the frequency by 2 each. The predividers are switched off when the switches are in the **UP** position. For best performace, the predividers should be turned off and $N$ set to 5 for a $\times 10$ divider. Thus, a 75MHz to 85MHz DDS signal is multiplied to 750MHz to 850MHz. The DDS can only synthesize up to 100MHz.



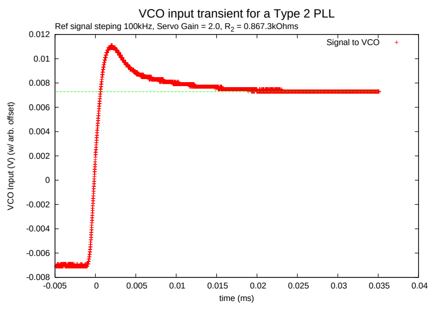
(a) Servo Gain = 0.5



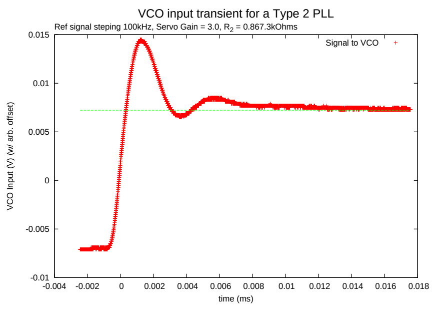
(b) Servo Gain = 1.0



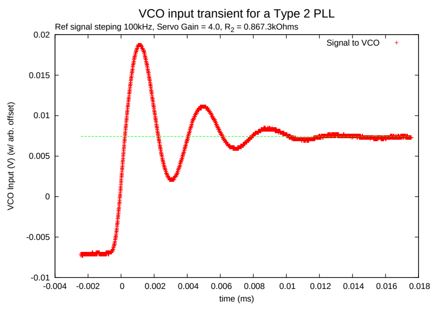
(c) Servo Gain = 1.5



(d) Servo Gain = 2.0



(e) Servo Gain = 3.0

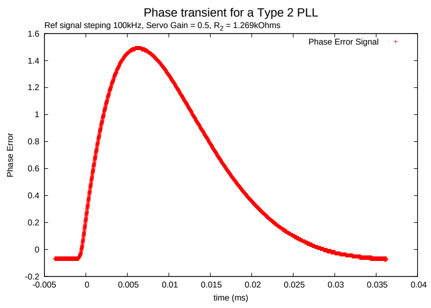


(f) Servo Gain = 4.0

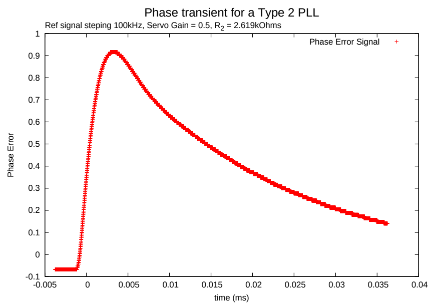
Figure 9.3: VCO voltage response to a frequency step as a function of phase detector gain. $\gamma = 0.7$ occurs when the transient above the final steady state extends approximately $\frac{1}{4}$ of the step between the steady states. This occurs at a gain of 1.5 here.

| | 750MHz at VCO (3.0GHz path) | 850MHz at VCO (6.8GHz path) |
|---|---|---|
| Gain Crossover Frequency | $301.3 \pm 0.7$ kHz | $306.2 \pm 0.7$ kHz |
| Gain | $607 \pm 18$kHz | $671 \pm 20$ kHz |
| Calculated damping constant $\gamma$ | 1.23 | 1.29 |
| Integrated phase noise, 100Hz to 20MHz | $8.0 \pm 0.8$ mrad | $9.2 \pm 1.0$ mrad |
| Servo Gain at crossover | $6.20 \pm .05$ | $5.59 \pm .02$ |

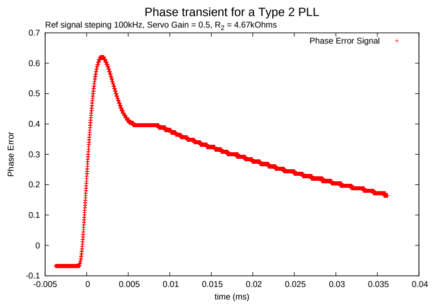
Table 9.2: Optimized RbSS system PLL specifications. Note that these figures are for response prior to frequency doubling. The signal frequency is 750-850MHz, and the reference frequency is 75-85MHz.



(a) Servo Gain = 0.5



(b) Servo Gain = 1.0



(c) Servo Gain = 1.5

Figure 9.4: Plots showing the phase error signal response to a frequency step as a function of $R_2$ which controls the Integrator:Proportional ratio of the PID loop filter. The loop becomes unstable at high integrator values.

# Chapter 10

# Frequency Multiplication

The 750MHz or 850MHz signal generated by the PLL is multiplied 4x or 8x to obtain 3.0GHz and 6.8GHz signals. To keep harmonics well outside the bandwidth of the bandpass filters, successive steps of doublers were used instead of a quad or 8x multiplier. The signal gain at each part of the chain is kept in check by appropriate amplifiers and attenuators. A solid state attenuator and a high isolation PiN diode are used to control the signal and power characteristics of the system. Part numbers and the multiplication chain are shown in 10.1.

| | |
|---|---|
| Multipliers | MC ZX90-2-11, -19, and -36 |
| Directional Coupler | MC ZX30-12-4 |
| Voltage Controlled Attenuator | Penstock PSAT 025-1 |
| PiN Diode | American Microwave SW-2000-1 |
| Bandpass Filters | Minicircuits VBFZ 1690, 3590, and 6260 |
| SPDT switch | MC ZFSWA-2-46 |
| Amplifier under 3GHz | Minicircuits ZX60-4016E |

Table 10.1: Parts for the RbSS system Multiplier chains. Layouts are shown in Figure 10.1

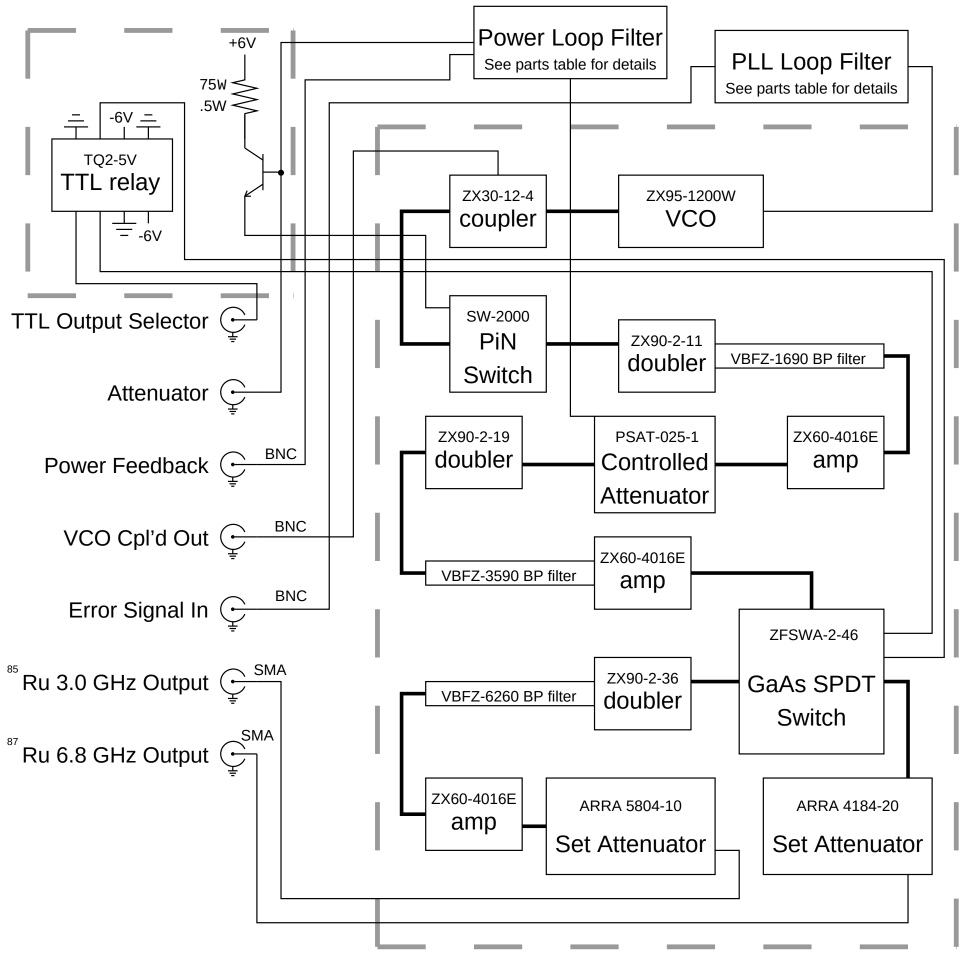


Figure 10.1: The first multiplication and amplification box.

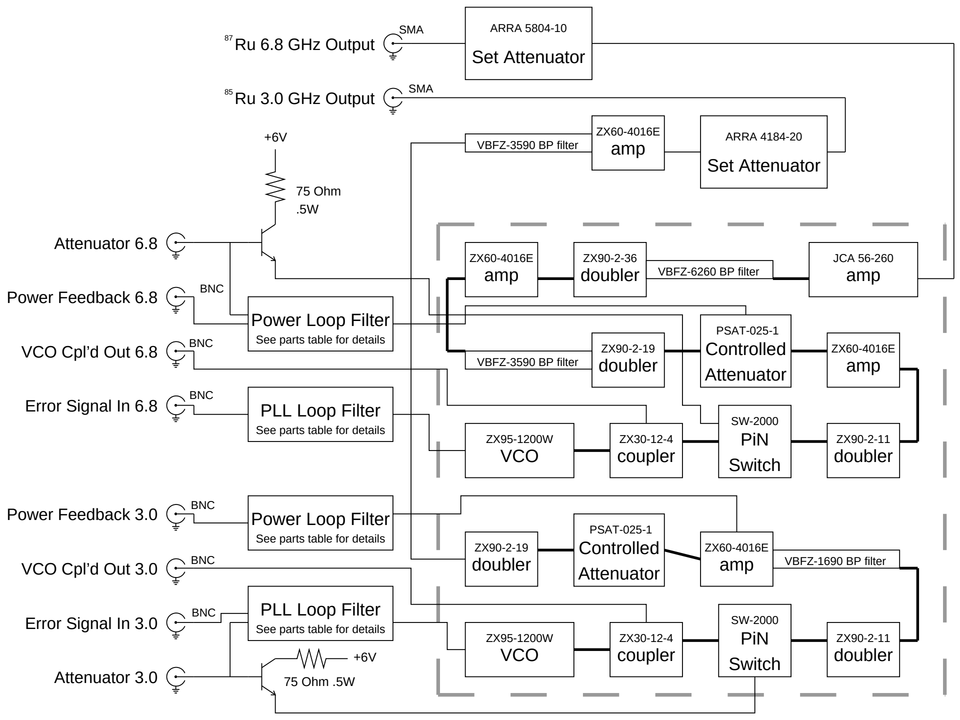


Figure 10.2: The second multiplication and amplification box.

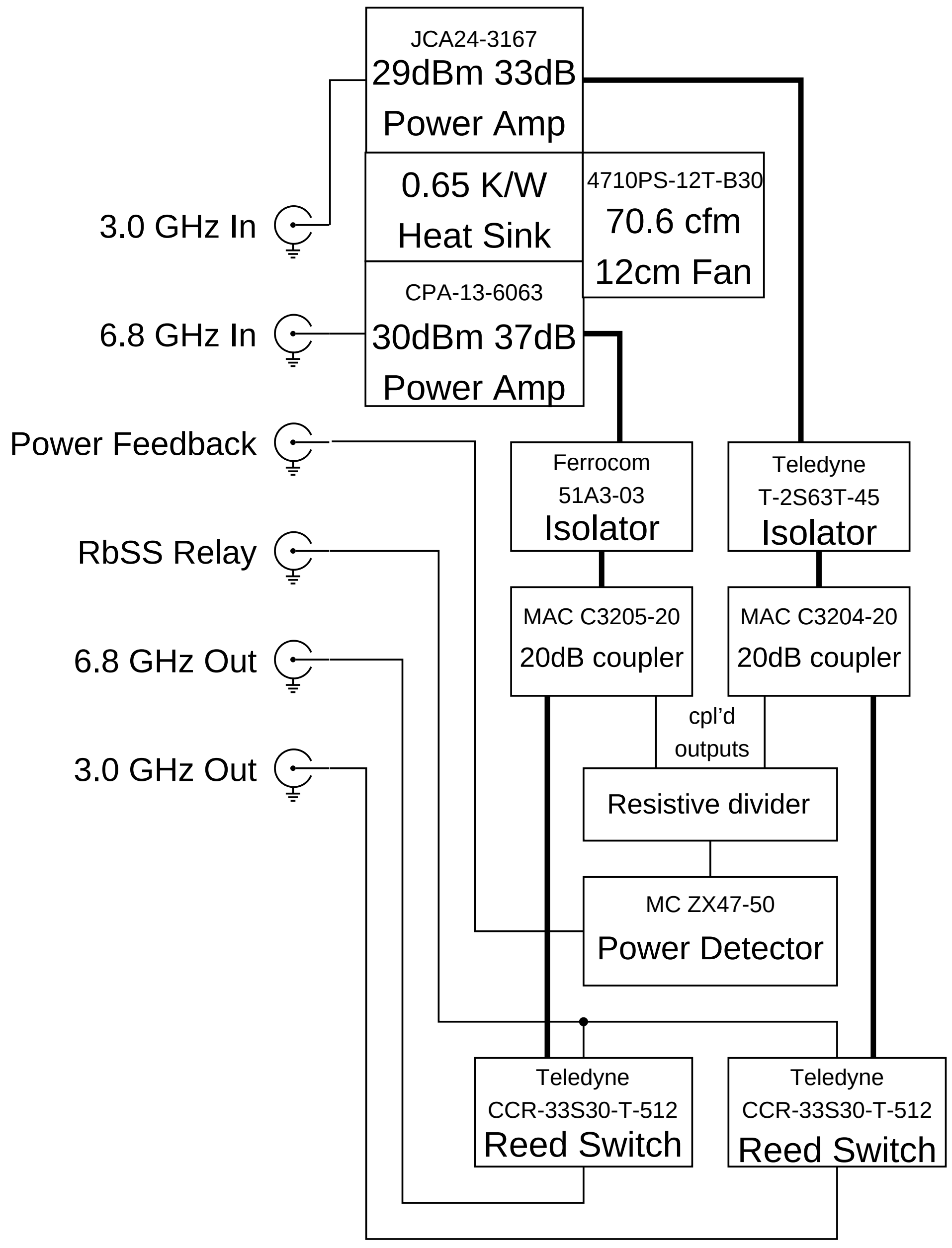


Figure 10.3: The first system's power amplification box.

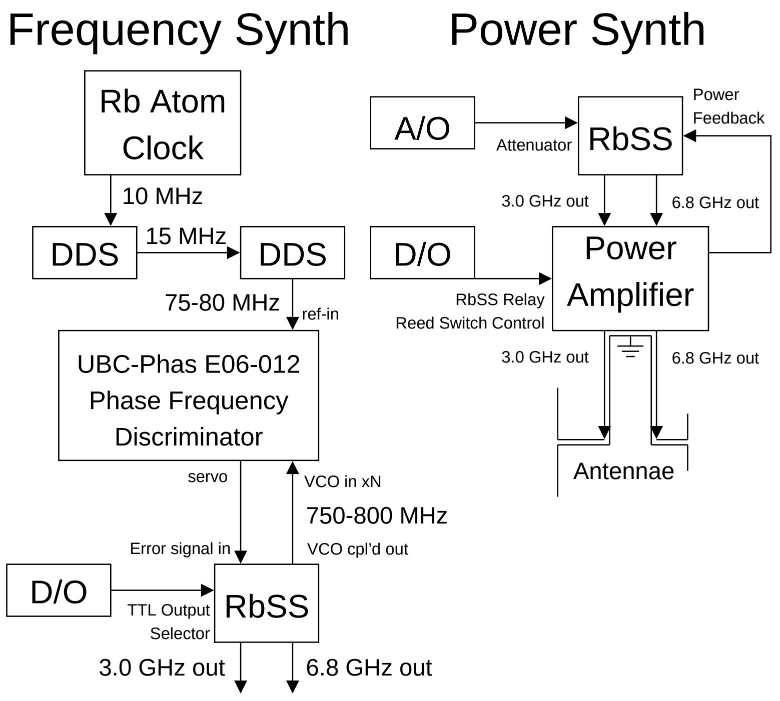


Figure 10.4: Layout of RbSS component systems.

# Chapter 11

# Amplification and Power Feedback

The final signals at both 3.0 and 6.8 GHz are amplified up to approximately 1W for transmission to the antennae. A schematic of the amplification chain is shown in 10.1. These final output power is measured and a fed back to the voltage controlled oscillator near the beginning of the doubling chain. The power feedback loop consists of the amplifier chain, a directional coupler to a solid state microwave power detector, a PI loop filter, and separate voltage controlled oscillators and PiN diodes early in the chain. A schematic of the loop is shown in 11.1. Terminated reed switches after the feedback loop control the final output to the antennae. The compression of the power amplifiers was measured in Figure 11.2

In order to scale the setpoint as the power output in dB and due to the logarithmic response of the VCA, the PID controller required an exponential amplifier in addition to the PID integrator. The maximum power output of the amplifiers, without railing the stabilization circuit is given in Table 11.2.

The power output of the system as a function of a shared voltage applied to the PiN diode and variable attenuator and as a function of the setpoint is given in Figures 11.3 and 11.4. The maximum stable power outputs of the system are given in Table 11.2

To calculate the power available at the dipole trap,

$$I = \frac{P \times G}{A} \tag{11.1}$$

$$= \frac{10^{27.43\mathrm{dBm}/10}\mathrm{mW} \times 10^{6.42\mathrm{dB}/10}}{4\pi(15\mathrm{cm})^2} \tag{11.2}$$

$$= 8.2 \pm 2.3\mathrm{W/m}^2 \tag{11.3}$$

where G is the antenna gain, and the distance to the atoms is estimated to be 15cm for the 3.0GHz antenna and 13cm for the 6.8GHz antenna.

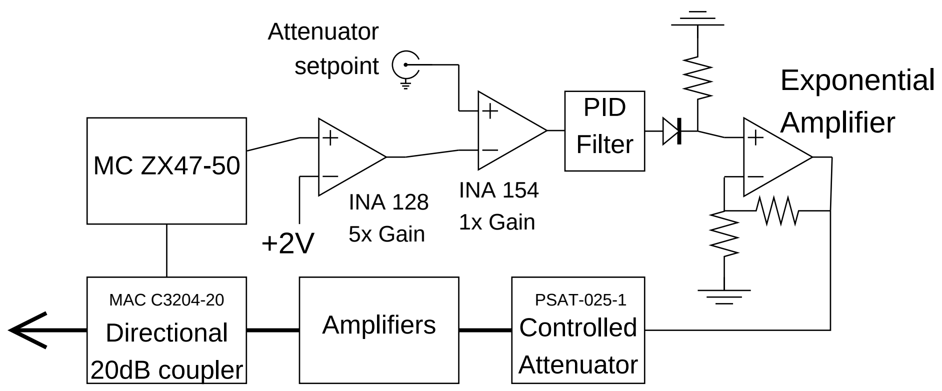


Figure 11.1: Schematic of the power feedback loop.

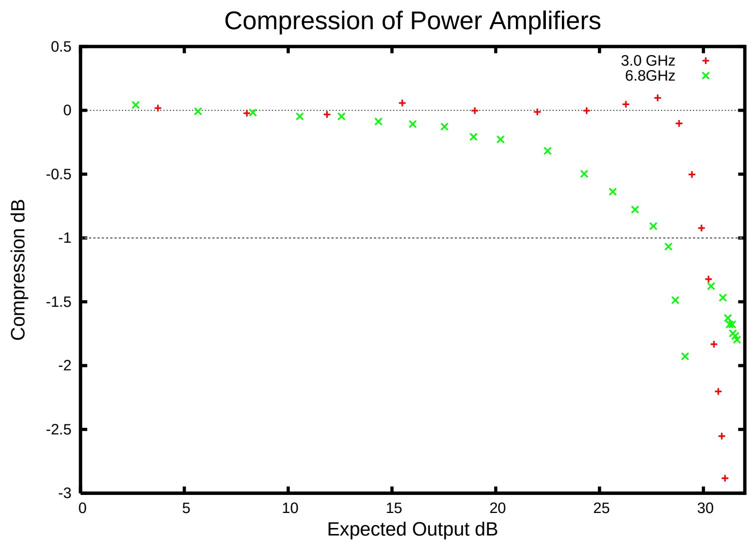


Figure 11.2: Power output compression (deviation from linearity) of the 6.8GHz and 3.0GHz power amplifiers.

| Component | Value |
|---|---|
| C2 | .01$\mu$F |
| R2, R18, R37, R39<br>R46, R50, R51, R57 | 0 Ω |
| R45 | 49.9Ω |
| R13, R41, R61 | 1kΩ |
| C3, R43, R44 | 3.83kΩ |
| R4, R42 | 5kΩ pot |
| R15 | 10kΩ |
| R10, +12V to U5 input | 20kΩ |
| R1 | 50kΩ pot |
| R16 | 100kΩ |
| R14 | diode 1N1418W |
| C2 | 0.01$\mu$F |
| R40 | 0.33$\mu$F |
| U1 | INA154U |
| U2, U5, U6 | LF356 or OP37G |
| U8 | INA128U |
| J3 | BNC jack |
| J1, J4, pwr in | Molex 70543 series |
| Decoupling Caps | |
| C12, C13, C20, C21 | 0.1$\mu$F |
| C10, C11, C22, C23 | 0.33$\mu$F |
| C41, C42 | 470$\mu$F |

Table 11.1: Power Stabilization loop filter component values. R42 and R1 are set to 2.05kΩ and 10.7 kΩ respectively. The optional voltage regulation is as in Table 9.1.

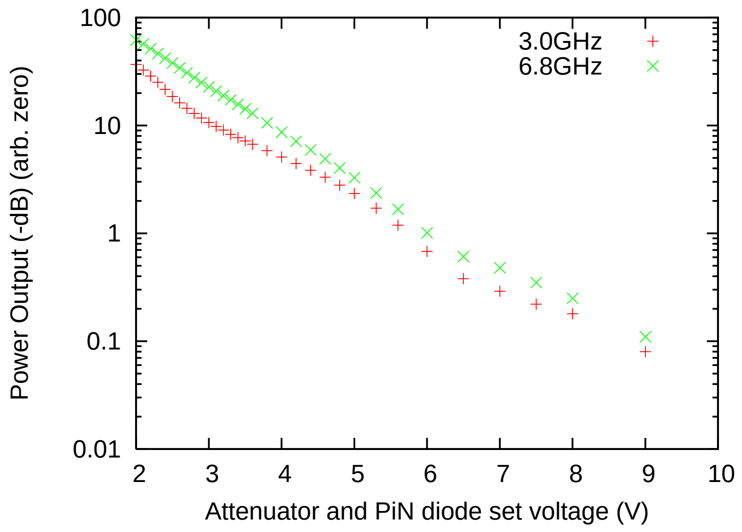


Figure 11.3: RbSS power output as a function of voltage applied to PiN diode and the variable attenuator.

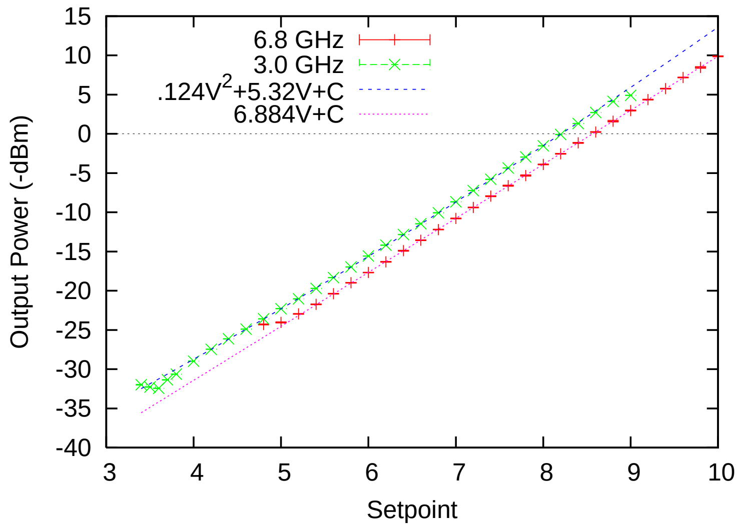


Figure 11.4: RbSS power output as a function power setpoint.

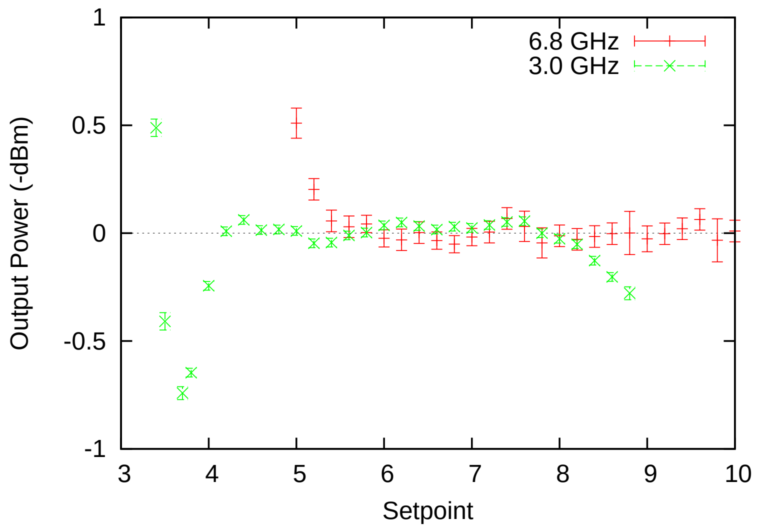


Figure 11.5: RbSS power output residuals as a function power setpoint.

| Location | Power |
|---|---|
| 3.0GHz | |
| Amplifier Box Output | $28.6 \pm 1.0$ dBm |
| Antenna Input | $27.43 \pm 1.25$ dBm |
| At the Dipole Trap | $8.2 \pm 2.3$ W/m$^2$ |
| 6.8GHz | |
| Amplifier Box Output | $27.7 \pm 0.7$ dBm |
| Antenna Input | $22.83 \pm 0.7$ dBm |
| At the Dipole Trap | $2.4 \pm 0.5$ W/m$^2$ |

Table 11.2: Maximum Stabilized Power output of the RbSS system.

# Chapter 12

# Antennae

To radiate the microwaves into the MOT cell, efficient and compact antennae were required. An existing RF system for controlling the hyperfine states of Lithium used simple loops of wire under the cell to couple to the atoms. Such a system would be inefficient for microwaves as we approach a far-field radiation pattern at the shorter wavelengths where inductive coupling fails. Several types of antennae were considered. High directivity and impedance matching were required. A half-wave dipole design was selected due to its ease of manufacture, calculability of the radiation pattern, and good impedance matching. A reflector place behind the antenna increases the directivity and tunes the impedance to 50Ω. The 6.8GHz antenna is shown in Figure 12.1.

This antenna was modeled using the NEC software package. The reflector was approximated by a series of wires behind the main antenna. A directivity plot was calculated and measured for the horizontal axis in front of the antenna as in Figure 12.4. The directivity measurements were taken before the installation of the RbSS power stabilization circuit, therefore the power output varies between taking data on either side of $\phi = 0$. However, the shape on either side matches the predicted gain reasonably.

Using directional couplers, the VSWR of the antennae were measured to be 1.03 for the 3.0GHz antenna, and 1.19 for the 6.8GHz antenna.

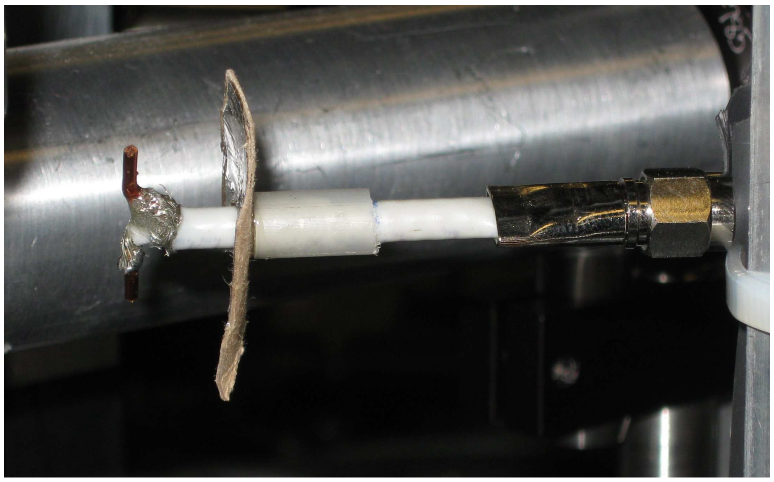


Figure 12.1: The 6.8GHz dipole antenna with reflector.

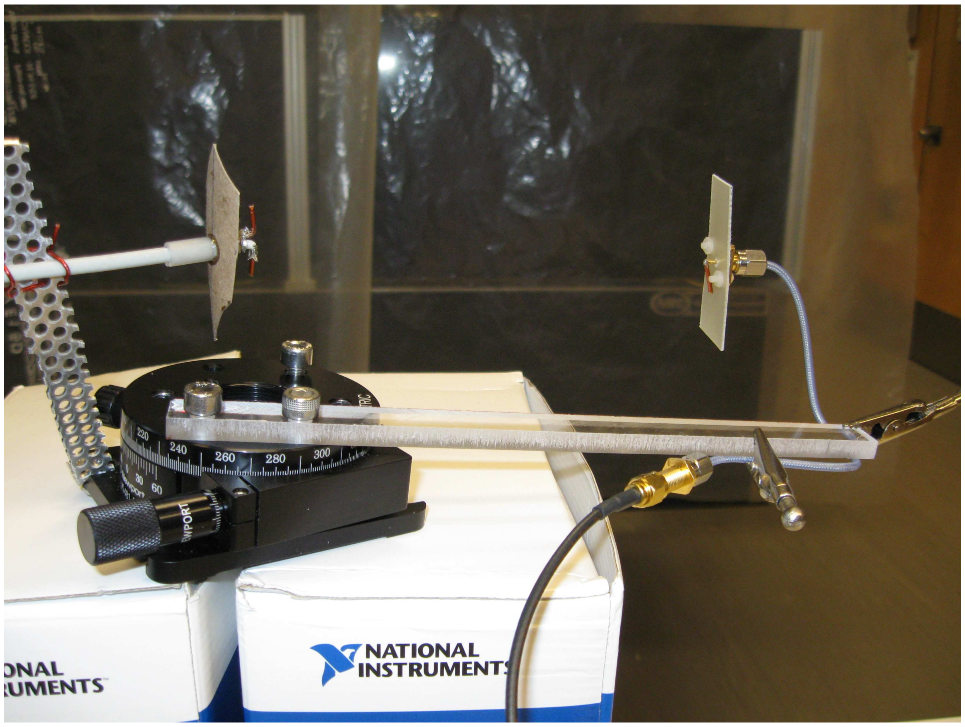
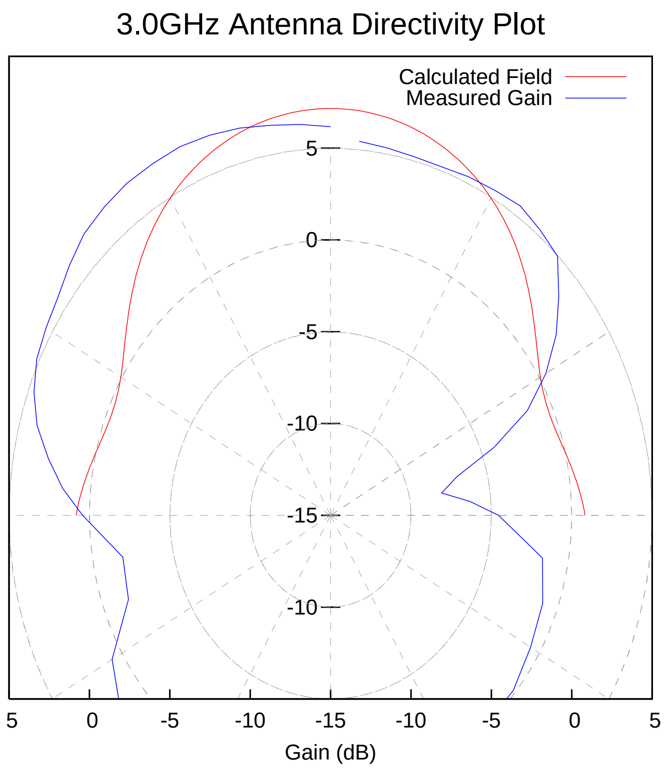


Figure 12.2: Measuring the antenna directivity.



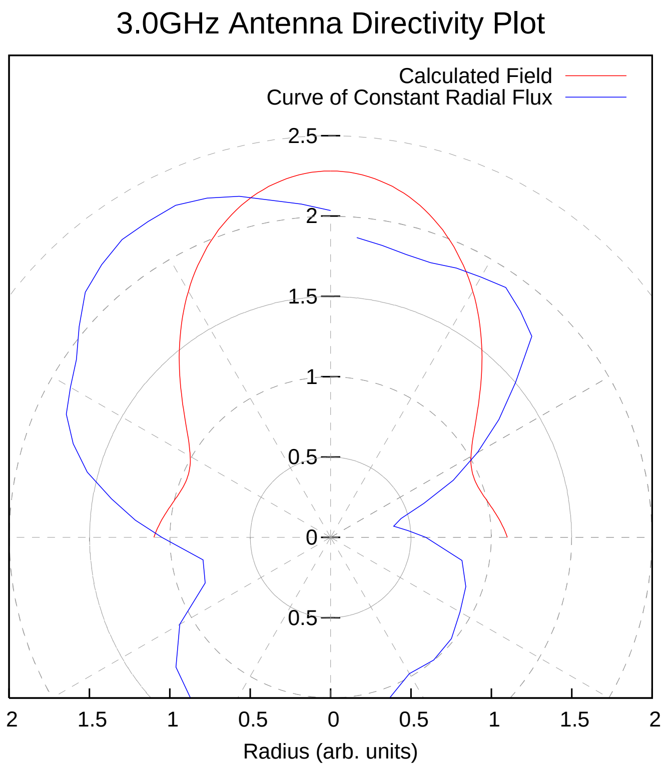
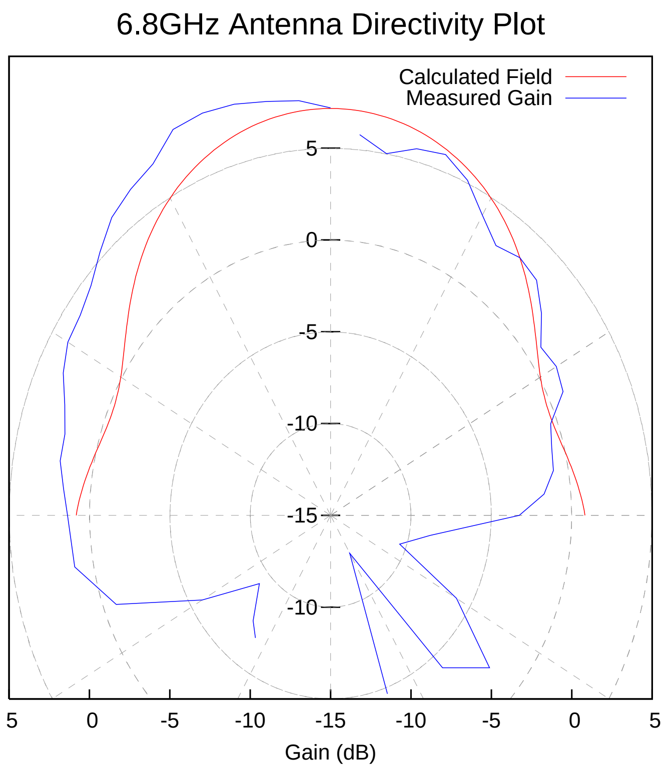


Figure 12.3: Directivity plots of the RbSS 3.0GHz antennae. The plot on the left shows the antenna gain while that on the right show the shape of the intensity distribution.



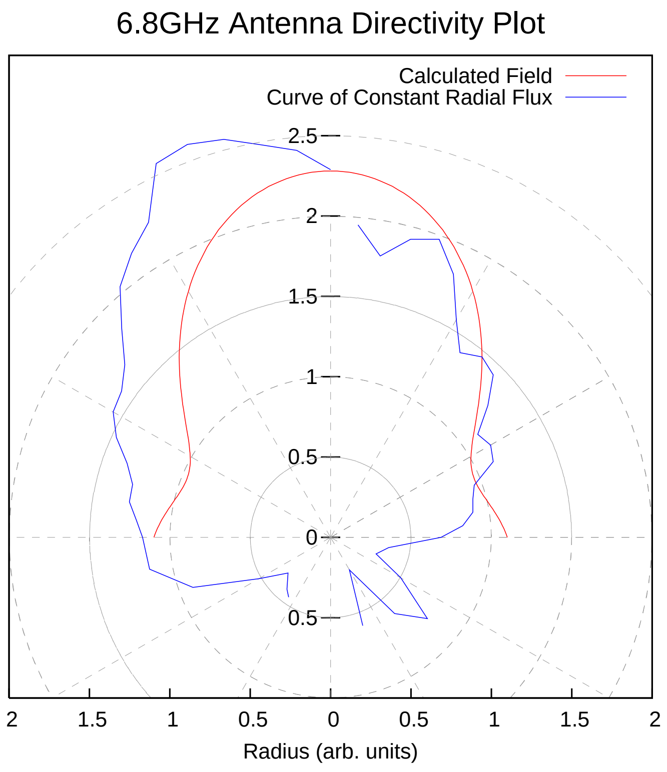


Figure 12.4: Directivity plots of the RbSS 6.8GHz antennae. The plot on the left shows the antenna gain while that on the right show the shape of the intensity distribution.

# Chapter 13

# RbSS Software

Python software to control the RbSS system is located in the MOLExperimentRecipe module, as in Appendix B. The `RbSS_ramp()` and `RbSS_single_tone()` methods implement similar methods on the UTBus direct digital synthesizers, DDSs. `RbSS_single_tone()` sets a specific frequency and power output on the RbSS system. When setting `switch = False`, the reed switch at the output of the system remains closed, allowing the power feedback circuit to stabilize.

Prior to running `RbSS_ramp()`, `RbSS_single_tone()` should be run to make sure system is set near the correct frequency and power setting. As the DDS switches to ramped mode, it briefly ($\sim 10\mu$s) turns off before starting. The `RbSS_ramp()` method waits long enough before ramping to restabilize the frequency.

The `_RbSS_set()` method is used by the other methods to convert an `ampl` value to a setpoint on the RbSS box. The functions fitted in Figure 11.4 are used to convert a dB value to a setpoint value. `ampl=1` sets the system at its maximum stable output, `ampl=0` sets the system at no output, and any negative number sets the output at that many dB below maximum.

# Bibliography

[1] *Apogee Instruments COM (ActiveX) API Specification*, April 2007.

[2] Cheng Chin, Rudolf Grimm, Paul Julienne, and Eite Tiesinga. Feshbach resonances in ultracold gases. preprint, Sept 2008.

[3] B. Deh, C.Marzok, C. Zimmermann, and Ph. W. Courteille. Feshbach resonances in mixtures of ultracold $^6$li and $^{87}$rb gases. *Phys. Rev. A*, 77(010701), Jan 2008.

[4] K. Ladouceur et al. A compact, laser cooling apparatus for simultaneous cooling of lithium and rubidium. preprint, 2008.

[5] Christopher J. Foot. *Atomic Physics.* Oxford University Press, 2005.

[6] Meng (Ray) Gao. Building a Dual-Species Optical Dipole Trap for Ultracold Lithium-6 and Rubidium-85,87. Master's thesis, University of British Columbia, Vancouver, BC, Canada, April 2008.

[7] Floyd M. Gardner. *Phaselock Techniques.* John Wiley & Sons, 3 edition, 2005.

[8] Michael E. Gehm. Properties of $^6$li. available at http://www.phy.duke.edu/research/photon/qoptics/techdocs/, February 2003.

[9] Will Gunton. The loading and storage of li and rb in an optical dipole trap. Master's thesis, University of British Columbia, Vancouver, BC, Canada, April 2009.

[10] K. M. Jones, E. Tiesinga, P. D. Lett, and P. S. Julienne. Photoassociation spectroscopy of ultracold atoms: long-range molecules and atomic scattering. *Rev. Mod. Phys.*, 78:483–535, 2006.

[11] SJM Kuppens, KL Corwin, KW Miller, TE Chupp, and CE Wieman. Loading an optical dipole trap. *PHYSICAL REVIEW A*, 62(1), JUL 2000.

[12] Keith Ladouceur. Experimental advances toward a compact dual-species laser cooling apparatus. Master's thesis, University of British Columbia, Vancouver, BC, Canada, April 2008.

[13] Z. Li, S. Singh, T. V. Tscherbul, and K. W. Madison. Feshbach resonances in ultracold $^{85}$rb-$^{87}$rb and $^{6}$li-$^{87}$rb mixtures. *Phys. Rev. A*, 78(022710), 2008.

[14] Todd P. Meyrath. Multipurpose analog pid controller. available at http://george.ph.utexas.edu/ meyrath/informal/PID.pdf, December 2005.

[15] L. Santos, G. V. Shlyapnikov, P. Zoller, and M. Lewenstein. Bose-einstein condensation in trapped dipolar gases. *Phys. Rev. Lett.*, 85(9):1791, August 2000.

[16] Daniel Adam Steck. Rubidium 87 d line data. available at http://steck.us/alkalidata, Sept 2001.

[17] Daniel Adam Steck. Rubidium 85 d line data. available at http://steck.us/alkalidata, May 2008.

[18] Curt Wittig. The landau-zener formula. *J. Phys. Chem. B*, 109(17):8428–8430, March 2005.

# Appendix A

# Photoassociation Hardware Interface Software

## A.1 PAHardware

```
"""
Tested Alan Robinson
v1.0 March 17, 2009
v1.1 March 19, 2009
"""
import socket
import sys
import time
import types
import re
from pylab import *

def debug(m):
    print m
    sys.stdout.flush()

class AddressError(Exception):
    def __init__(self, instrument):
        self.value = instrument
    def str__(self):
        return "Failed to connect to %s" % self.value

class PAHardware(object):

    EOM = '\r\n'
    connected = dict()
    def __init__(self,host = "172.16.1.101",port = 15000,address = 1):
        self.timeout=5
        self.host = host
        self.address = address
        # self.scope_address = address
        self.s = self.__connect(port)
        self.set_Enut_addr(address)

    def connect(self, port = 15000):
        self.s = self.__connect(port)

    def __connect(self, port = 15000):
        if self.host in PAHardware.connected: return
PAHardware.connected[self.host][0]
        s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        s.settimeout(self.timeout)
        try:
            #debug( 'trying to connect,host = %s,port = %s'%(host,port))
            s.connect((self.host, port))
            self.server_id = s.getpeername()
            start_time = time.time()
            self.connection_timestamp = time.strftime("%y%m%d_%H%M%S")
            #debug( 'connected to %s at %s' %
                #(self.server_id,self.connection_timestamp))
        except:
            raise
            mesg='connection error,  '+str( sys.exc_info()[0])
            debug(mesg)
        PAHardware.connected[self.host] = [s, -1] # Add host to dictionary.
                #GPIB address unset.
        return s

    def close(self): # May close other instances of PAHardware on the same host!
        if not self.host in PAHardware.connected: return
        self.s.close()
        del PAHardware.connected[self.host]
        mesg='connection to %s closed'%str(self.server_id)
        #debug(mesg)
        return mesg

    def send_request(self, request, checkaddr=True):
        if checkaddr:
            if not PAHardware.connected[self.host][1] == self.address:
                self.set_Enut_addr(self.address)
        request += self.EOM
        bytes = self.s.send(request)
        if bytes < len(request):
            debug('ERROR: %d of %d bytes sent'%(bytes,request))
            raise Exception('send error')

    def enut_request(self,request):
        request += self.EOM
        bytes = self.s.send(request)
        if bytes < len(request):
            debug('ERROR: %d of %d bytes sent'%(bytes,request))
            raise Exception('send error')

    def get_reply(self,bytes_to_read='all',verbose=False):
        data=''
        term=0
        tstart=time.time()
        if bytes_to_read.lower() == 'all':
            term  = 1
            bytes_remaining = 10**6
        else:
            bytes_remaining = bytes_to_read
        if verbose: debug( "bytes to read = %s" %bytes_to_read)
        while 1:    #if bytes_to_read is OK then read them
            if bytes_remaining>0:
                try:
                    new_data = self.s.recv(bytes_remaining)
                except:
                    debug ('error reading bytes from server')
                    break
            else:
                new_data=''
            L=len(new_data)
            if verbose: debug( "read '%s' bytes" %L)
            data+=new_data
            bytes_remaining -= L
            #print repr(data[-1]), bytes_remaining
            if bytes_remaining <=0:
                if verbose: debug( "%s of required %s bytes"
%(len(data),bytes_to_read))
                break #read required number of bytes --> exit loop
            time_sofar=time.time()-tstart
            if time_sofar>=self.timeout:
                debug( 'timeout')
                if verbose: debug( "read %s of required %s bytes"
%(len(data),bytes_to_read))
                break  #timed out --> exit loop
            if term == 1:
                bytes_remaining=10**6
                if data[-1]=='\n':
                    if verbose: debug( "Done. Read all %s bytes" %(len(data)))
                    break
        self.last_read = len(data)
        return data

    def ident(self, instrument): # Raises error if not connected to instrument.
                #Makes sure that commands don't mess up other GPIB instruments.
        self.send_request('*IDN?')
        ident = self.get_reply()
        if ident.find(instrument) == -1:
            raise AddressError(instrument)

    #Commands for Ethernut#
    def get_Enut_version(self):
        self.enut_request('!VERSION')
        return self.get_reply()

    def get_Enut_addr(self):
        self.enut_request('!getiaddr')
        return self.get_reply()

    def set_Enut_addr(self, address):
        self.enut_request('!setiaddr %d' % address)
        PAHardware.connected[self.host][1] = address
```

## A.2 HP8663Asyn

```
"""
Tested by Alan Robinson
February 8, 2009
v 1.0 March 17, 2009
v 1.1 March 19, 2009
"""
import sys
import time
from PAHardware_v1 import PAHardware

class HP8663Asyn(PAHardware):  # Interface for HP8663A Synthesizer through
# PAHardware enut interface.  All commands are also accessible from the front
# panel.

    UnitDict = dict( dBm = 'DM',
                      mV = 'MV',
                      uV = 'UV',
                      dB = 'DB',
                      Hz = 'HZ',
                     kHz = 'KZ',
                     MHz = 'MZ',
                     GHz = 'GZ',
                 percent = 'PC'
                   )

    def __init__(self,host = "172.16.1.101",port = 15000,address = 20):
        PAHardware.__init__(self, host, port, address)
        self.set_ren()

    def close(self):
        self.set_ren(0)
        PAHardware.close(self)

    # generic set functions

    def reset(self):
        self.send_request('SP 00')
    def set_amplitude(self, value=-30., unit='dBm'):
        self.send_request('AP %.3f %s' %(value, self.UnitDict[unit]))

    def set_off(self):
        self.send_request('AO')

    def set_freq(self, value=100., unit='MHz'):
        self.send_request( 'FR %f %s' % (value, self.UnitDict[unit]))

    # ##automatic sweep functions###

    def set_sweep(self, mode='off'):
        SweepDict = dict( off = 'W1',
                         cont = 'W2',
                       single = 'W4'
                        )
        self.send_request('T2') # set 1ms per step.
        self.send_request(SweepDict[mode])

    def set_sweep_step(self, step=0.1, unit='Hz'): # Rate is this per ms.
# Minimum 0.1Hz below 640MHz, 0.2Hz above.
        self.send_request('N3 %f %s' % (step, self.UnitDict[unit]))

    def set_sweep_span(self, span=0., unit='MHz'): # Use in conjunction with
# set_frequency for the centre frequency
        self.send_request('FS %f %s' % (span, self.UnitDict[unit]))

    def set_sweep_range(self, start=100., stop=100., unit='MHz'):
        self.send_request('FA %f %s' % (start, self.UnitDict[unit]))
        self.send_request('FB %f %s' % (stop, self.UnitDict[unit]))

    # externally controlled stepping.  GPIB controls each step, one at a time.

    def set_incr(self, step=0.1, unit='Hz'):
        self.send_request('FR IS %f %s' % (step, self.UnitDict[unit]))

    def walk(self, direction = 1):
        if direction == 1:
            self.send_request('FR UP')
        elif direction == -1:
            self.send_request('FR DN')

    #Commands for Ethernut#

    def set_ren(self, mode = 1):
        """enable(1)/disable(0) REN line (pin 17 of GPIB)"""
        self.enut_request('!SETREN %d' % mode)

def main(): # Example of use.
    synth = HP8663Asyn() # Initializes interface

    ### Single frequency setup
    synth.set_freq(100., 'MHz')
    synth.set_amplitude(0, 'dBm')

    ### Ramp gradually to a new frequency
    synth.set_sweep_range(100., 120., 'MHz')
    synth.set_sweep_step(1, 'kHz') # Rate 1000kHz/s
    synth.set_sweep('single')
    time.sleep(21)  # Wait for sweep to finish, 20s
    synth.set_freq(120., 'MHz') # Switch mode back to single frequency

    ### Software control each step.  Use single frequency mode.
    synth.set_incr(10, 'Hz')
    for i in range(0,10):
        synth.walk(1)  # Increment frequency 10Hz at a time
    for i in range(0,10):
        synth.walk(-1) # Decrement frequency 10Hz at a time

if __name__ == "__main__":
    main()
```

## A.3 AgilentCounter

```
"""
Tested Alan Robinson
v0.0 February 8, 2009
v0.1 February 16, 2009
v1.0 March 17, 2009
```

```
v1.1 March 19, 2009

Methods to use:

set_default(self) Reset the counter to factory defaults
set_timer(self, time_ms=10) Set gate timer for frequency measurements
run(self, measurement) Sets the counter to take continuous measurements and sets
measurement to report.
get_measurments(self, n=1) Get measurements defined in run() in a float array
"""
import sys
import time
from PAHardware_v1 import PAHardware

class AgilentCounter(PAHardware): # Implements PAHardware GPIB interface for
Agilent Counter.

    def __init__(self,host = "172.16.1.101",port = 15000,address = 8):
        PAHardware.__init__(self, host, port, address)
        self.ident()
        self.timer = 1000

    def ident(self): # Raises error if not connected to instrument.  Makes sure
# that commands don’t mess up other GPIB instruments.
        PAHardware.ident(self, ’53132A’)

    def set_default(self): # Reset counter and set impedance to 50 Ohms
        self.ident()
        self.send_request(’*RST;*CLS;*SRE 0;*ESE 0;:STAT:PRES;:INP1:IMP 50;
:INP2:IMP 50; ROSC:EXT:CHEC OFF; ROSC:SOUR EXT’)

    def set_timer(self, time_ms=100): # Set gate timer for frequency
# measurements
        self.timer = time_ms
        self.send_request(’:FREQ:ARM:STAR:SOUR IMM; :FREQ:ARM:STOP:SOUR TIM;
:FREQ:ARM:STOP:TIM %d MS’ % self.timer)

    def run(self, measurement = ’FREQ 1’): # starts measurements as identified
# in Agilent counter manual.
        self.send_request(’:FUNC \"%s\"; :INIT:CONT ON’ % measurement)
        time.sleep(self.timer/1000. + 0.1) # Wait for first measurement

    def get_wait(self): # Returns the recommended wait time between
# get_measurement() commands
        return self.timer/1000. + 0.1

    def get_measurements(self, n=1): # Takes at least 100ms between measurements
        data = []
        count = time.clock()
        for x in range (0, n):
            # Wait for next measurement
            while time.clock() <= count + x*(self.timer+100)/1000.:
                pass
            self.send_request(':DATA?')
            tidbit = float(self.get_reply().rstrip('/n'))
            # print tidbit
            data.append(tidbit)
        return data

def main(): # Example of use.  Make sure you have two signals before starting.
    cntr = AgilentCounter() # Initializes counter
    cntr.set_timer(100) # Sets Counter gate timer in ms
    # Starts continuous measurements and waits for the first measurement
    cntr.run('FREQ 1')
    # Get value of first measurement in Hz. Returns a list of values.
    # First value selected.
    freq1 = cntr.get_measurements(1)[0]
    cntr.run('FREQ 2')
    freq2 = cntr.get_measurements(1)[0]
    print "Ch1: %gHz Ch2: %gHz" % (freq1, freq2)

    cntr.run('FREQ 1')
    freq1 = cntr.get_measurements(1)[0]
    freq2 = cntr.get_measurements(1)[0]
    time.sleep(cntr.get_wait())
    freq3 = cntr.get_measurements(1)[0]
    print "%gHz is probably the same as %gHz but is different than %gHz" %
(freq1, freq2, freq3)

if __name__ == "__main__":
    main()
```

## A.4 Wavemeter

```
###
# Version 1.0
# Alan Robinson
# February 16, 2008
# Connects to Bristol Wavemeter over USB interface using CLDevIface.dll
# proprietary library
# All methods require reinitialization of the USB interface due to the
# unreliability of the USB connection.
# Todo: 1) Make quiet, 2) Verify that that a timeout error did not occur (failed
# measurement)
###
import os
import sys
import time
path_components = ['E:', 'UTBus_Recipes', 'Work', 'Rb State Selector',
'PyWavemeter', 'CodeGenerator']
path_components2 = ['C:', 'Program Files', 'Bristol Wavemeter']
sys.path.insert(0,os.sep.join(path_components))

from ctypes import *
# this module is found in the code generator folder
from Types import *

CCD_DATA_LEN = 1024
#
MAX_K = 32
#
NUM_FIFO_DWORDS = 6
#
RESPONS_TBL_LEN = 32
#
VERS_LEN = 8
#
RESPONS_ENTRIES = 16
#
```

```
RESPONS_WAVE = 0
#
RESPONS_VAL = 1
######################################################################
######################################################################

class WavemeterException(Exception):
    def __init__(self,error_code):
        msg = "Error code %d" % (error_code)
        Exception.__init__(self,msg)

######################################################################
######################################################################
class Wavemeter(object):
    def __init__(self):
        self.dll = cdll.CLDevIface
        self.device = -1
        self.port = 3
        self.set_medium(0)
        self.set_power_units('dBm')
        self.set_lambda_units('nm')
    def close(self):
        self.dll.CLCloseDevice.argtypes = [c_int]
        ret = self.dll.CLCloseDevice(self.device)
        #if ret != 0:
            #raise WavemeterException(ret)
    def get_lambda(self):
        self.dll.CLGetLambdaReading.argtypes = [c_int]
        self.dll.CLGetLambdaReading.restype = c_double
        self.open(self.port)
        ret = self.dll.CLGetLambdaReading(self.device)
        #if ret == -1.:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
        return ret
    def get_power(self):
        self.dll.CLGetPowerReading.argtypes = [c_int]
        self.dll.CLGetPowerReading.restype = c_float
        self.open(self.port)
        ret = self.dll.CLGetPowerReading(self.device)
        #if ret == -1.:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
        return ret
    def open(self, port):
        self.dll.CLOpenUSBSerialDevice.argtypes = [c_int]
        self.device = self.dll.CLOpenUSBSerialDevice(port)
        #if self.device == -1:
            #raise WavemeterException(self.device)
    def set_average(self, enable = 0, value = 10): # 0 - disable; 1 - enable
        self.dll.CLSetAverageEnable.argtypes = [c_int, c_byte]
        self.open(self.port)
        ret = self.dll.CLSetAverageEnable(self.device, enable)
        #if ret != 0:
            #raise WavemeterException(ret)
        self.dll.CLSetAverageValue.argtypes = [c_int, c_uint]
        ret = self.dll.CLSetAverageValue(self.device, value)
        #if ret != 0:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
    def set_lambda_units(self, units = 'nm'):
        unitDict = dict( nm = 0,
                        GHz = 1,
                         cm = 2
                        )
        self.dll.CLSetLambdaUnits.argtypes = [c_int, c_uint]
        self.open(self.port)
        ret = self.dll.CLSetLambdaUnits(self.device, unitDict[units])
        #if ret != 0:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
    def set_medium(self, medium = 0): # 0 - vacuum; 1 - air
        self.dll.CLSetMedium.argtypes = [c_int, c_uint]
        self.open(self.port)
        ret = self.dll.CLSetMedium(self.device, medium)
        #if ret != 0:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
    def set_power_units(self, units = 'dBm'):
        unitDict = dict( mW = 0,
                        dBm = 1
                        )
        self.dll.CLSetPowerUnits.argtypes = [c_int, c_uint]
        self.open(self.port)
        ret = self.dll.CLSetPowerUnits(self.device, unitDict[units])
        #if ret != 0:
            #raise WavemeterException(ret)
        self.close()
        time.sleep(0.02)
```

# Appendix B

# RbSS software

```
def _RbSS_set(self, chan = 1, ampl = 0): # sets switches and voltage controlled
attenuator

    self.RbSS_switch(0)
    self.RbSS_FSK(0)
    self.RbSS_setpoint.set_scaled_value(0)
    if chan == 1:
        self.RbSS_sel(0) # Sets 3.0 GHz path
        if ampl == 1:
            ampl = 4
        elif ampl < 0:
            if ampl < -30:
                ampl = -30
                print "Requested RbSS power output exceeded minimum.  Power set
to minimum, -30dB."
            ampl = ((-ampl+79.6)/.124)**0.5 - 21.45
        else:
           if ampl != 0:
               print "Invalid RbSS setpoint.  Power off."
           ampl = 0
    elif chan == 2:
        self.RbSS_sel(1) # Sets 6.8GHz path
        if ampl == 1:
            ampl = 5.2
        elif ampl < 0:
            if ampl < -32.84:
                ampl = -32.84
                print "Requested RbSS power output exceeded minimum.  Power set\
to minimum, -32.84dB."
            ampl = (-ampl+36)/6.884
        else:
            if ampl != 0:
                print "Invalid RbSS setpoint. Power off."
            ampl = 0
    else:
        self.RbSS_sel(0)
    self.RbSS_setpoint.set_scaled_value(ampl) # Sets VCA and PiN switch

def RbSS_on(self, on = 1):
    self.RbSS_switch(on)

# sets single tone signal to prepare loop.
def RbSS_single_tone(self, freq, ampl = 0, switch = False):

    self.RbSS_ref.reset()
    self.RbSS_ref.set_amplitude(0.7)
    if freq < 5*ghz: # Check if 3 or 6.8GHz path is to be used
        chan = 1
        self.RbSS_ref.single_tone(freq / D['phiDetN'] / 8)
    else:
        chan = 2
        self.RbSS_ref.single_tone(freq / D['phiDetN'] / 16)

    if not ampl == 0:
        self._RbSS_set(chan, ampl)
        # wait for frequency to stabilize.  Takes care of worst case.
        self.wait_ms(100)
        if switch:
            self.RbSS_switch(1)
    else:
        self._RbSS_set()

def RbSS_ramp(self, f1, fdelta, rate, ampl = 1): # Set rampable signal
# controlled by digital output.  Prepare loop using RbSStone before using this
# method.

    f1 /= D['phiDetN'] * 8
    fdelta /= D['phiDetN'] * 8
    rate /= D['phiDetN'] * 8
    chan = 1

    if max(f1, f1+fdelta) > 100*MHz: # Check if 6.8GHz path is to be used
        f1 /= 2
        fdelta /= 2
        rate /= 2
        chan = 2

    self.RbSS_FSK(0)
    self.RbSS_ref.ramped_FSK(f1, f1+fdelta, fdelta*1./rate)
    self.RbSS_ref.set_amplitude(0.7)

    self._RbSS_set(chan, ampl)
    self.wait_ms(5) # wait for loop to stabilize after DDS transient.
    if not ampl==0:
        self.RbSS_switch(1)
    self.wait_ms(0.1) # wait for switch to stabilize
    self.RbSS_FSK(1)
    self.wait_s(fdelta*1./rate+0.001)
    self._RbSS_set(chan, 0)
    self.RbSS_FSK(0)
```