Radiofrequency Synthesis System for Laser Modulation

Jiajun Shi

Faculty Advisor: Professor David A. Hanneke
December 22, 2014

Submitted to the
Department of Physics and Astronomy of Amherst College
in partial fulfillment of the
requirements for the degree of
Bachelor of Arts with honors.

Abstract

Doppler cooling and resolved sideband cooling enable us to bring a trapped ion to its motional ground state, creating the possibility of performing quantum logic spectroscopy on the ion. Doppler cooling requires a laser beam resonant with a transition between the ion's internal states. Sideband cooling, meanwhile, needs a pair of laser beams detuned from resonance. For effective cooling, we need a pulse sequence in which each pulse has programmable frequency, amplitude, phase, and duration to modulate the laser beam to our needs in each cooling scheme.

This thesis describes the construction of a versatile two-part radiofrequency synthesis system for laser modulation. One part of the system involves a voltage-controlled oscillator in conjunction with a phase-locked loop. We use it to generate a 6.6GHz signal to make a resonant sideband. The pulse sequence system relies on a more agile frequency source, a direct digital synthesizer. Using a field-programmable gate array to control multiple direct digital synthesizers simultaneously, we achieved fast switching among eight frequency+amplitude+phase profiles by installing external transistor-transistor logic triggers to control the pulse widths and switch the output profiles.

Acknowledgments

Thank you, Professor Hanneke. Your guidance and resourcefulness have made the process of this thesis project the most enjoyable and intellectually rewarding academic experience I have ever had. The realm of electrical engineering is not an easy place for beginners to venture into, yet you helped me with no prior knowledge understand it. Thank you for always being a great mentor, and sharing your knowledge and thoughts as a physicist. It has been a great year working with you.

I would also like to thank all the great physics professors. Although I have not taken any classes with many of you, you have always been willing to help and offer your advice. I was late to get on board, but I never felt any difficulty to blend in. I feel a sense of affiliation in this department, and for that, I am very grateful.

To Jim Kubasek and Norm Page, thank you for helping me with the box enclosure, which is a crucial part in the system construction. Without your help, I would not be able to do this alone.

To my fellow physics students, I have enjoyed every moment working together with you. We may have been facing the toughest challenges in this college, yet we all passed through them together. Some of my good friends in the department have already graduated, and here I wish you all embark on journeys to realize your dreams.

To all my friends not majoring in physics, thank you for your continuous supports in my toughest year in college. You have made my senior year a good year I will never forget.

Here is a special thanks to Vlad Negnevitsky– although I do not personally know you, I feel grateful for all the supports on hardware control you provided. I wish your research goes well in the future.

This work was supported by the National Science Foundation, the Research Corporation for Science Advancement, and the Amherst College Dean of the Faculty.

Finally, I am grateful for the supports from my family. Thank you, mom and dad, for having faith in me and encouraging me whenever I feel down. Your love is what carries me through all the hardships in my life.

Contents

List of Figures

Chapter 1

Introduction

The endeavor to unify the fundamental interactions into one single field has not reached a conclusion yet. Many unified field theories, such as superstring or Kaluza-Klein model, have prompted us to seek evidence of many of their predictions. One such prediction is the time-variation of the proton-electron mass ratio $\mu = m_p/m_e$, which, if present, could provide strong evidence to support certain unified field theories and to refute others [3]. More importantly, non-zero $d\mu/dt$ would change the way we view fundamental constants, and help us better understand the fundamental interactions.

Measurement of $d\mu/dt$ will however require superb precision, given that the time variation would be hardly observable. Molecules are good candidates to perform such measurements on, because they have internal transitions sensitive to μ. For a diatomic molecule, the energy difference between two rotational energy levels scales with the m_e/M, where the reduced mass M of the nuclei is dependent on the proton mass m_p. By measuring the transition energies of two rotational transitions of the molecule, we can obtain a frequency different sensitive to the changes in the ratio μ.

The method to perform this precision measurement is quantum logic spectroscopy, the application of laser pulses to study the internal structures of microscopic particles. As we hope to find μ, we realize the difficulty to perform it on a molecule because precision measurement of transition energies requires the atomic or molecular species to have a narrow reference transition, effective state preparation/detection, and possibility to be cooled down to minimize frequency fluctuations [4]. Molecules satisfy only the first requirement. We can however perform state manipulations on an atomic ion, which satisfies the requirements for spectroscopy due to its rather simple structures, and transfer these actions to a more complicated molecular ion coupled with the the atomic ion to form normal modes of motion through Coulomb interactions. The atomic ion, called a logic ion, and the molecular ion, called a spectroscopy ion, have to be coupled by Coulomb interactions when they are co-trapped inside a linear Paul trap. The Coulomb interactions allow the logic ion to sympathetically cool the spectroscopy ion to its motional ground state. We can then manipulate the internal states of the spectroscopy ion to find the target transition frequency. This method allows us to circumvent the complex structures of the molecular ion that impose a difficulty on the study of the molecule's μ-sensitive transitions by taking advantages of the simple and better known structures of the logic ion. The spectroscopy ion and the logic ion we will use are ${}^{16}O_2^+$ and ${}^{9}Be^+$, respectively.

Two laser cooling schemes, Doppler cooling and resolved sideband cooling, will be used for effective cooling. The laser beams in both methods need to be modulated at certain stages. This thesis focuses on the construction of a versatile radiofrequency (RF) synthesis system for laser modulation. In Chapter 2, I provide the theories behind the cooling schemes, and discuss their implications on the requirements for the RF synthesis system. In Chapter 3, I present the process of system construction, and the ways to control the system. The actual performance of the system is shown in Chapter 4, which also includes the discussion of the potential improvements that can be applied to the system.

Chapter 2

Theory

Laser cooling provides a fundamental tool in quantum spectroscopy on charged particles. In our set-up, the cooling process occurs in a linear Paul trap designed by Shenglan Qiao [5]. After co-trapping a spectroscopy ion and a logic ion, diatomic and atomic respectively, we apply laser cooling techniques to bring both ions to their motional ground state. The actual cooling process aims to cool only the logic ion, but this equivalently cools the the normal modes of motion arising from the coupling of the two ions via the Coulomb interaction. The cooling process is necessary in quantum spectroscopy because measuring the transition energy between the two rotational states we are interested in requires both the spectroscopy ion and the logic ion to be in their motional ground state, so that when we drive the target transition, we can know the effectiveness of the drive from the two ion's normal modes of motion, given that we prepare both ions in their motional ground state and appropriate internal states.

Doppler cooling is the first step in our cooling scheme. Efficient in bringing the temperature of the ions down significantly, it is however incapable of lowering the temperature below a certain barrier called the Doppler limit. This is where resolved sideband cooling, which can lower the temperature further beyond the Doppler limit and eventually reach the motional ground state, takes over. Resolved sideband cooling involves coupling of two of the ion's internal states with motional states, performing Raman transition between two coupled states, and repumping. Resolved sideband cooling needs to be repeated several times with different sets of laser beams before the ground state is reached. It is worth noting that the requirement for fast changing of laser frequency in sideband cooling serves as the primary motivation for us to develop a versatile laser modulation system.

2.1 Beryllium Ion

^{9}Be$^+$ is a good candidate for the logic ion in quantum logic spectroscopy not only because it has a simple structure, but also the nuclear spin $I = 3/2$ it has couples with the spin $S = 1/2$ of the valence electron in the magnetic field created by the nucleus. This coupling gives rise to hyperfine structures with large splittings. The ground state of the ion splits into eight hyperfine states in this structure. The first excited state has fine structures, $P_{1/2}$ and $P_{3/2}$ due to spin-orbit coupling, and also hyperfine structures. $P_{3/2}$ has 16 hyperfine states, while $P_{1/2}$, which is separated from $P_{3/2}$ by 197GHz in the fine structure, has eight hyperfine states. See Fig 2.1 for a diagram showing the energy levels of ^{9}Be$^+$. Later on we will denote $S_{1/2}|F = 2, m_F = 2\rangle$ as $|{\downarrow}\rangle$, and $S_{1/2}|F = 1, m_F = 1\rangle$ as $|{\uparrow}\rangle$, as the transition between those two is crucial to resolved sideband cooling discussed in Section 2.3. [6, p. 13]

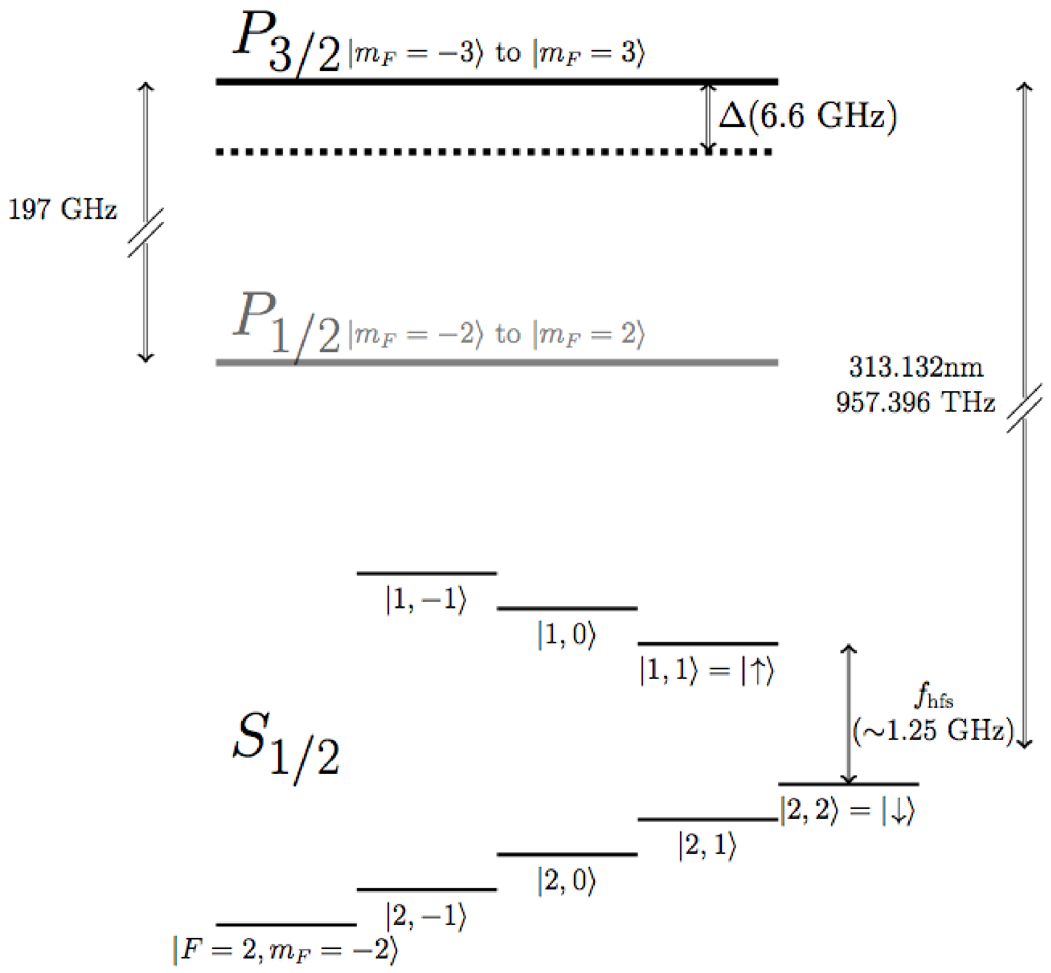


Figure 2.1: Energy level diagram of $^9\mathrm{Be}^+$. The hyperfine splittings of the $S = 1/2$ state are shown. Notice that we can safely neglect $P_{1/2}$ when transitions between $S_{1/2}$ and $P_{3/2}$ are involved because the splitting between $P_{1/2}$ and $P_{3/2}$ is very large (197GHz). [7]

2.2 Doppler Cooling

Doppler cooling is the first step in ground-state cooling. It uses a laser beam to cool the logic ion down, rather than heating it up. This may sound impossible at beginning, but the internal structure of the ion allows this to happen. Suppose the logic ion is trapped in our linear Paul trap, behaving like a three-dimensional simple harmonic oscillator, we would then like to constrain its motions in all three directions. It is possible to use a single laser beam to achieve this, but we have to make sure the direction of the laser propagation has components along all three axes, thus meaning that the laser must not be launched along any of the Paul trap's axes of symmetry.

As its name suggests, Doppler cooling uses the Doppler effect to damp the atom's motion. The laser frequency that the ion perceives is higher, or blue shifted, if the ion moves towards the laser source. If the shifted laser frequency corresponding to a certain velocity of the atom becomes resonant with an electronic transition in the atom, the logic ion then absorbs a photon and receives a momentum kick in the direction of the laser propagation. The now-excited atom spontaneously emits a photon in a random direction, and receives a momentum gain in a direction opposite to the emitted photon. The randomness of the direction of this subsequent momentum gain guarantees that the momentum gain due to spontaneous emission averages out over time. We see that if the ion is moving towards the laser source, its motion will be damped by the radiative force, which causes the atom's kinetic energy to decrease, equivalently lowering its temperature. [8, p. 267-268]

One thing to keep in mind is that if the ion is moving away from the light source, it sees a lower laser frequency, making it possible for the ion to absorb photons whose perceived frequency is resonant. This results in a gain in kinetic energy, equivalent to a heating process. To prevent this, we detune the laser frequency below resonance, in which case more photons can be absorbed by the oscillating ion moving towards the light source and less by the ion moving away. Red detuning of the laser frequency would then result in a net loss of the ion's kinetic energy, effectively cooling the ion down.

2.2.1 Doppler Limit

Spontaneous emission grants the ion random-direction momentum gains that average out over time. It seems that manipulating the laser detuning to make the ion undergo continues directional absorptions and random emissions would cool it down to absolute zero. However, this is not the case. There is a non-zero standard deviation to the momentum of the ion due to the spontaneous emissions. The random momentum kicks make the atom engage in three dimensional random walk, which is equivalent to a heating process. This hinderance imposes a limit to which the atom can be cooled by the Doppler beam, and it is called the recoil temperature, defined as $T_{Recoil} = \frac{\hbar^2 k^2}{2mk_B}$ [9, p. 65], where k is the wavenumber of the light.

We see that in order to perform Doppler cooling repeatedly, we would like to use two of the ion's internal states to perform a cycling transition. Our starting point is the $S_{1/2}|F = 2, m_F = 2\rangle$ state. At this point, we have a distribution among the hyperfine manifold of the ground state since any excited state decays fast. The frequency detuning makes sure all the states in the manifold are visible to the Doppler cooling beam. [7] Starting from here, we use circularly polarized (σ^+) Doppler beams to excite the ion. The selection rule governs $\Delta m_F = +1$ during the excitation process. This drives the transition from $S_{1/2}|2, 2\rangle$ to $P_{3/2}|3, 3\rangle$. Once the ion is at $P_{3/2}|3, 3\rangle$, the only state the ion can decay to is $S_{1/2}|2,2\rangle$, since the selection rule only allows $\Delta m_F = \pm 1$ or 0 in this transition, and we know that no ground states have $m_F = 3$ or 4. Hence, the transition between these two states is unique, meaning that no other transition will happen during the cooling process. The capacity of Doppler cooling is not only limited by atom recoiling, but also the broad natural linewidth– we call it Γ– that corresponds to the electronic transition used in the cooling scheme. The linewidth of the $P_{3/2}|3,3\rangle$ is 19.4MHz. This linewidth imposes a temperature limit $T_{Doppler} = \hbar\Gamma/2k_B$ because it sets a limit to the cooling rate, and it is usually much higher than the previously mentioned recoil temperature. [8, p. 267] Hence, as the linewidth limit is the lowest temperature de facto in this cooling scheme, it is called the Doppler limit to address the extent to which the atom can be cooled by the Doppler method.

At the Doppler limit, we have an ensemble of motional states. In this Boltzmann distribution, we can find each motional state's population by

$$P(n) = \frac{e^{-E_n/kT}}{Z} = \frac{e^{-nh\nu/kT}}{Z} \tag{2.1}$$

where T is the Doppler limit obtained from $T_{Doppler} = \hbar\Gamma/2k_B$. Z is the partition function of the ensemble, and ν is the trap frequency (1MHz). We should know that the next cooling scheme needs a motional state to begin with, so we ought to find a motional state below which we have most of the probabilistic distribution of states, or population of states, meaning that any higher motional state has negligible population. For instance, we may find the first motional state, say $|n\rangle$, to have less than 1/1000 of the total population by using Eq 2.1, then begin with the sideband

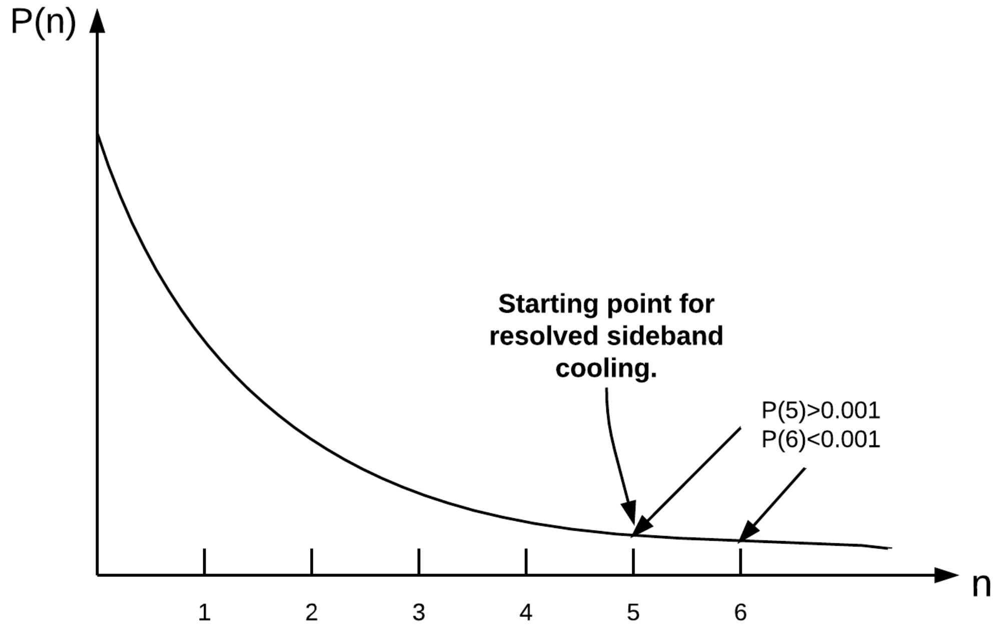


Figure 2.2: Boltzmann distribution at the Doppler limit. In this limit, the states beyond n=5 have negligible populations. We begin resolved sideband cooling at n=5 motional state.

cooling on $|n-1\rangle$. Fig 2.2 shows a simple example for $n = 6$.

The Doppler limit prompts us to seek a more versatile cooling method that operates below the Doppler temperature. Resolved sideband cooling, based on the theory of Raman transition, is one such cooling scheme that helps us bring the ion further down below the Doppler limit to the lowest energy levels.

2.3 Resolved Sideband Cooling

When we reach the Doppler limit, the temperature at that point gives rise to a Boltzmann distribution of the motional states. These motional states are associated with the 1MHz trap system, a harmonic oscillator. Starting from this ensemble of various motional states, we apply a step-by-step cooling scheme called resolved sideband cooling to decrease the ion's temperature beyond the Doppler limit.

2.3.1 Stimulated Raman Transitions

To begin with, we first couple two internal states of the ion mentioned in Section 2.1, $|\uparrow\rangle$ and $|\downarrow\rangle$ for shorthand, and make transitions between the two. The energy splitting between the two states is 1.25GHz, as seen in Fig 2.1. When we couple these hyperfine levels, we make sure $|\downarrow\rangle$ is coupled with a vibrational state, say $|n\rangle$, while $|\uparrow\rangle$ couples with $|n-1\rangle$. The transitions between these two states are called Raman transitions, the core process in sideband cooling. [10]

Driving the Raman transition requires two laser beams. First we find the resonance frequencies corresponding to the transitions from the target states, $|\downarrow, n\rangle$ and $|\uparrow, n-1\rangle$, to a higher transient energy state, $P_{3/2}$, then apply a detuning to the two frequencies. These detuned frequencies are no longer resonant with the transitions, but their difference is at resonance with the splitting between the two couple states, 1.25GHz. Those will be the laser frequencies we use for our two beams that will make the transition happen. When engaging in Raman transition, the ion's internal states oscillate at a frequency, known as the Rabi frequency. It would then be possible to make all population land on the $|\uparrow, n-1\rangle$ state if we time our beams accurately. The Raman beams that make the population end up exclusively on $|\uparrow\rangle$ is called a π-pulse.

The Rabi frequency is given by

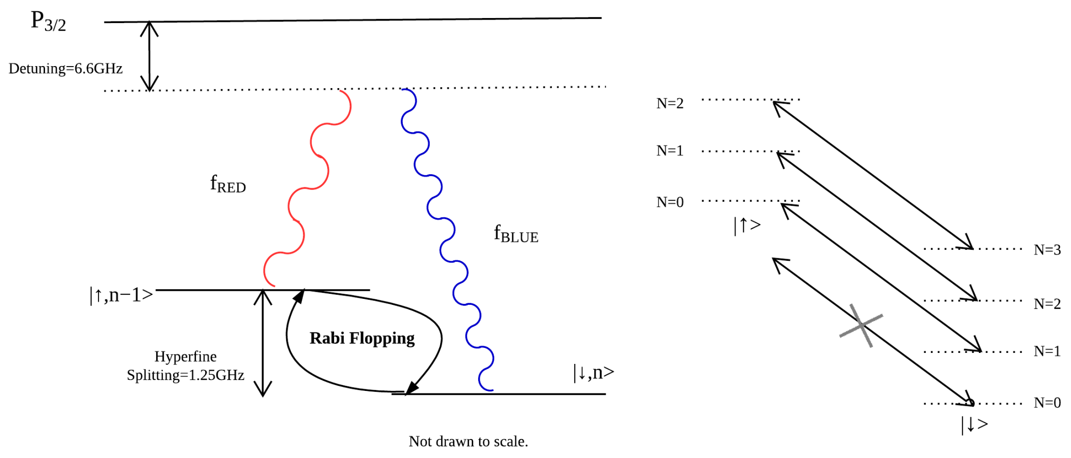


Figure 2.3: Left: Stimulated Raman transition between the two coupled states through a third transient state. f_{RED} and f_{BLUE} are the frequencies of the laser beams. Right: Example of Raman transition between low-n vibrational states.

$$\Omega_{n,n+1} = \Omega_{n+1,n} = \Omega_0 |\langle n+1| e^{i\eta(a+a^\dagger)} |n\rangle| = \Omega_0 e^{-\eta^2/2} \eta^1 \sqrt{1/n} L_n^1(\eta^2) \tag{2.2}$$

where Ω_0 is the interaction strength that can be obtained by finding the strength of the coupling between the two states. η is the Lamb-Dicke parameter for the coupling. [11] L_n^1 is the generalized Laguerre polynomial that equals

$$L_n^1(X) = \sum_{m=0}^{n} (-1)^m \binom{n+1}{n-m} \frac{X^m}{m!} \tag{2.3}$$

We can see that Rabi frequency is inversely proportional to the square root of n, which is associated with the state with higher energy among the two states. This means we need to use multiple sets of laser beams with different durations for different sets of coupled states, as they have different n values. If we use a particular π-pulse, say for $|\downarrow, n\rangle$ and $|\uparrow, n-1\rangle$, only $|\uparrow, n-1\rangle$ survives between the two, but not all $|\downarrow, n-1\rangle$ population turns into $|\uparrow, n-2\rangle$ due to Rabi frequency's dependence on the n value. Further cooling from this point requires new sets of Raman beams to couple two new states. Before the next stage can be initiated we will need a technique called laser repumping to ready the ion for the next Raman transition.

2.3.2 Repumping

After the Raman transitions, we removed a quantum from $|n\rangle$, but the $|n-1\rangle$ now couples with $|\uparrow\rangle$, preventing us from stimulating the Raman transition to remove a quanta from $|n-1\rangle$. To proceed, we need to apply an intermediate state manipulation technique called repumping.

Laser repumping allows for a transition from one internal state to another without changing the motional state. To explain this procedure, we need to break the short-hand and use explicit expressions for states. Let us assume sideband cooling leaves us at a motional state $|n\rangle$, with populations scattered around $S_{1/2}|F=2, m_F=2\rangle$ and $S_{1/2}|F=1, m_F=1\rangle$. We would then wish to bring all the population in $S_{1/2}|F=1, m_F=1\rangle$ to $S_{1/2}|F=2, m_F=2\rangle$ as complete as possible. Keep in mind that the following procedure will not involve any change in the motional state.

The first step in repumping is exciting the ion to a $P_{3/2}$ state with $m_F=2$, either $P_{3/2}|F=3, m_F=2\rangle$ or $P_{3/2}|F=2, m_F=2\rangle$, using a repumping beam that is resonant with either of the two transitions (same transition frequency), and also has the correct polarization to make $\Delta m_F=+1$. The excited ion then quickly decays into 3 possible states: $S_{1/2}|F=2, m_F=2\rangle$, $S_{1/2}|F=1, m_F=1\rangle$, or $S_{1/2}|F=2, m_F=1\rangle$ according to the selection rule. $S_{1/2}|F=2, m_F=2\rangle$ is the desired destination, and the population that decays into it will stay there because the the off-resonance repumping beam does not excite it. If the ion decays back to $S_{1/2}|F = 1, m_F = 1\rangle$, the repumping beam will pump it up again. The problem lies in the $S_{1/2}|F = 2, m_F = 1\rangle$ state– some population will stay here and undergo no more transitions. To tackle this, we need to apply a special technique– nuclear magnetic resonance (NMR)– to eliminate the population in this undesired state.

To proceed with NMR, we apply an external magnetic field varying at 1.25GHz, equal to the splitting between $S_{1/2}|F = 1, m_F = 1\rangle$ and $S_{1/2}|F = 2, m_F = 1\rangle$. Similar to Raman transitions, this induces a flopping between $S_{1/2}|F = 1, m_F = 1\rangle$ and $S_{1/2}|F = 2, m_F = 1\rangle$ at Rabi frequency. By applying the field for the duration that creates a π-pulse, we can remove all the population in $S_{1/2}|F = 2, m_F = 1\rangle$ state. We we are getting now is a distribution among $S_{1/2}|F = 1, m_F = 1\rangle$ and $S_{1/2}|F = 2, m_F = 2\rangle$. So, with repumping and NMR, we have successfully transferred part of the population in $S_{1/2}|F = 1, m_F = 1\rangle$ to $S_{1/2}|F = 2, m_F = 2\rangle$ without changing anything else. The population in $S_{1/2}|F = 2, m_F = 2\rangle$ can only increase in this stage, because nothing is exciting it. We will then repeat this whole process again and again, until we have almost all the state population in $S_{1/2}|F = 2, m_F = 2\rangle$, then perform sideband cooling for $|n - 1\rangle$ motional state. See Fig 2.4 for a diagram showing the transitions in the repumping process.

Repeated executions of stimulated transitions and repumpings can bring the ion down to its motional ground state, provided we find the correct starting point for sideband cooling from the Boltzmann distribution at the Doppler limit, and apply accurate Raman beams, repumping beams, and NMRs. We would then reach the endpoint of the complete cooling scheme. We will now present some implications

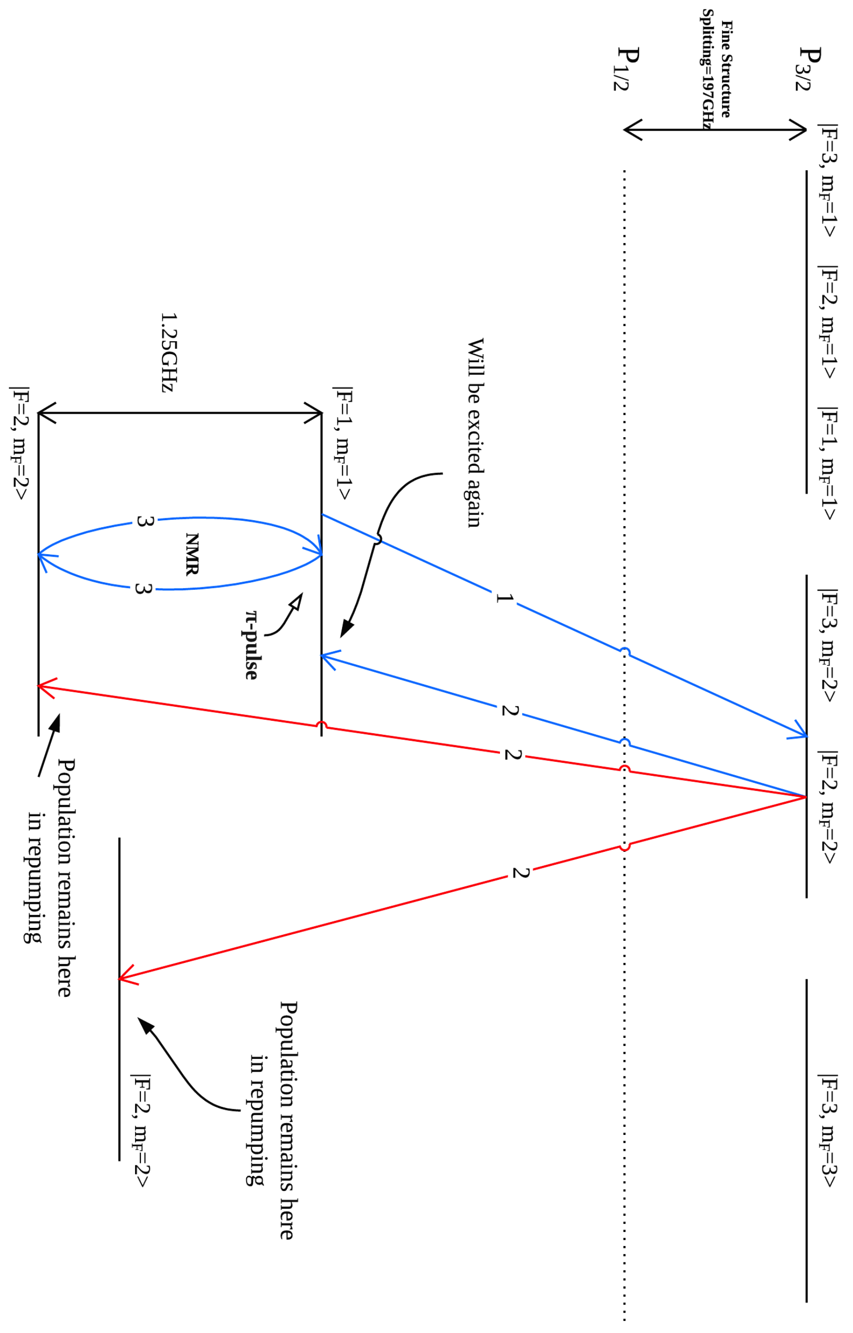


Figure 2.4: The process of the laser repumping and the nuclear magnetic resonance. The numbers on the transition lines indicate the order of that process taking place. Not drawn to scale.

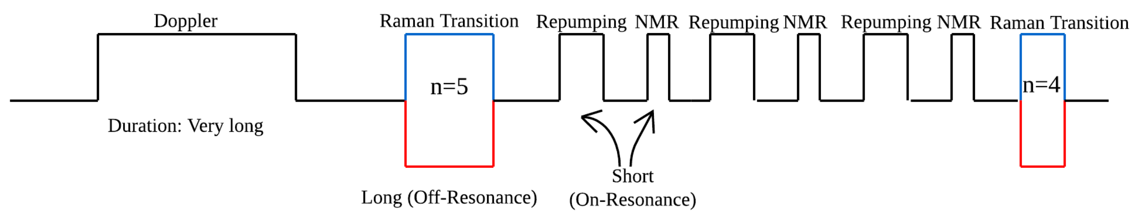


Figure 2.5: An example of the modulation pulse sequence. The two pulses for the Raman transition are shown both ways to indicate they are applied simultaneously. Three applications of the repumping and the NMR are shown, but there could be more for effective state preparation.

from this chapter on the required sequence of laser pulses.

2.4 Laser Pulse Sequence

It is important to think about the steps we need to go through to cool the ion down to its motional ground state. We first apply the Doppler beam with the detuning to decrease the ion's temperature to the Doppler limit, then choose a motional state that can address the upper limit for nearly all the motional states, and start using resolved sideband cooling to remove a quantum from this state. Two Raman beams forming a π-pulse will be used in the sideband cooling. When this is done, we are unable to remove another quantum. We have to apply the repumping beam and the NMR technique repeatedly to prepare the ion for another stage of sideband cooling, and proceed from there. See Fig 2.5 for a qualitative illustration of the laser pulse sequence we will need.

Knowing the requirements for the laser pulse sequence, we seek to build a versatile laser modulation system based on these requirements. In the next chapter, we will see how the theories behind the cooling schemes we discussed about prompted us to select qualified instruments for laser modulation accordingly.

Chapter 3

System

Performing quantum spectroscopy on ions requires precise manipulation of the laser frequency and pulse duration for effective cooling. With our frequency-locked external cavity diode laser and an acousto-optic modulator (AOM), we achieve this goal by constructing a versatile RF synthesis system capable of generating arbitrary pulse sequences. Our experiment demands full controllability on each pulse's frequency, amplitude, and phase such that we can modulate the laser beam in any way we want via AOM. We also want to be able to jump from one programmable pulse to the next quickly to avoid atom thermalization and be able to cool multiple modes of vibration.

Our laser is detuned from resonance by 6.6GHz. This detuned beam suppresses spontaneous emissions during the Raman transitions. In order to use the same laser for resonant transitions– as in Doppler cooling and repumping– we frequency modulate the beam to add a weak sideband at resonance.

In this chapter, we first demonstrate the process of controlling a detuning generator based on a voltage-controlled oscillator (VCO) evaluation board. Cheyenne Teng

[7] shows that the detuning works at 6.6GHz with its power at 20dBm. Without an option to modulate the signal amplitude on our oscillator, we apply amplifiers and attenuator to tune the power to the desired value. Frequency and phase stabilization is guaranteed by an on-board phase-locked loop (PLL) which utilizes a negative feedback mechanism to lock the VCO output to a 10MHz clock signal, so that we can detune the laser without disturbing its frequency stabilization. We then move on to introducing the construction of the pulse sequence system, which plays the most important role in laser modulation. Instead of using a VCO and PLL combination, we base our system on a direct digital synthesizer (DDS) to meet our requirements on the pulse sequences. Multiple channels of modulation signals are necessary in the cooling scheme, and while we employ multiple DDSes, we also need a field-programmable gate array (FPGA) as the central control unit to allocate the outputs and deliver commands. We can control 4 DDSes on an IC board with a FPGA on the same board using a Python script and a TTL pulse sequencing system as output trigger. With a backplane that supports up to 8 boards, our system is capable of achieving a maximum of 32 output channels, each with programmable frequency, amplitude, phase, and pulse duration.

3.1 Detuning

In resolved sideband cooling, we need a 6.6GHz detuning from $P_{3/2}$, a relatively large detuning compared to the linewidth of $P_{3/2}|3,3\rangle$ state. This way, by tuning our laser to the sideband of$P_{3/2}|3,3\rangle$, we can reduce the rate of spontaneous emission as those will hinder the cooling process by introducing additional kinetic energy to the ion, and more importantly, the coupling between the two states involved in the Raman transition will be stopped and the Rabi flopping no longer exists.

We use a laser beam whose frequency is already detuned by 6.6GHz from resonance. We will also use this laser for repumping, so a portion of the laser beam's power has to be resonant. This is done by modulating the laser using injection current modulation with a 6.6GHz RF signal to create two sidebands that are both 6.6GHz away from the laser frequency. One of the sidebands will be resonant, and makes a portion of the laser's power become the repumping beam we need. In sideband cooling, we do not want resonant light, so we will use a switch to turn off the RF signal input to eliminate the resonant sideband.

Our task is the generation of the 6.6GHz RF signal. 6.6GHz is a relatively high frequency– it falls in the label of microwave in radio spectrum. We will present a frequency generation device that does this job– voltage-controlled oscillator.

3.1.1 Voltage-Controlled Oscillator

A voltage-controlled oscillator (VCO) is an electronic instrument that oscillates at a frequency determined by the voltage input. It is an example of an LC circuit responding to a control voltage by varying its oscillation frequency. A stable DC input will result in a sinusoidal output signal whose frequency is instantaneously controlled by the input. A VCO is very useful in generating high frequency, although low frequency applications are also common. It is worth noting that one outstanding feature of a VCO is its low phase noise. An ideal oscillator would generate a perfect sine wave, revealing one line in the frequency spectrum. Realistic oscillators can never achieve such perfection, and will always introduce fluctuations in phase, resulting in

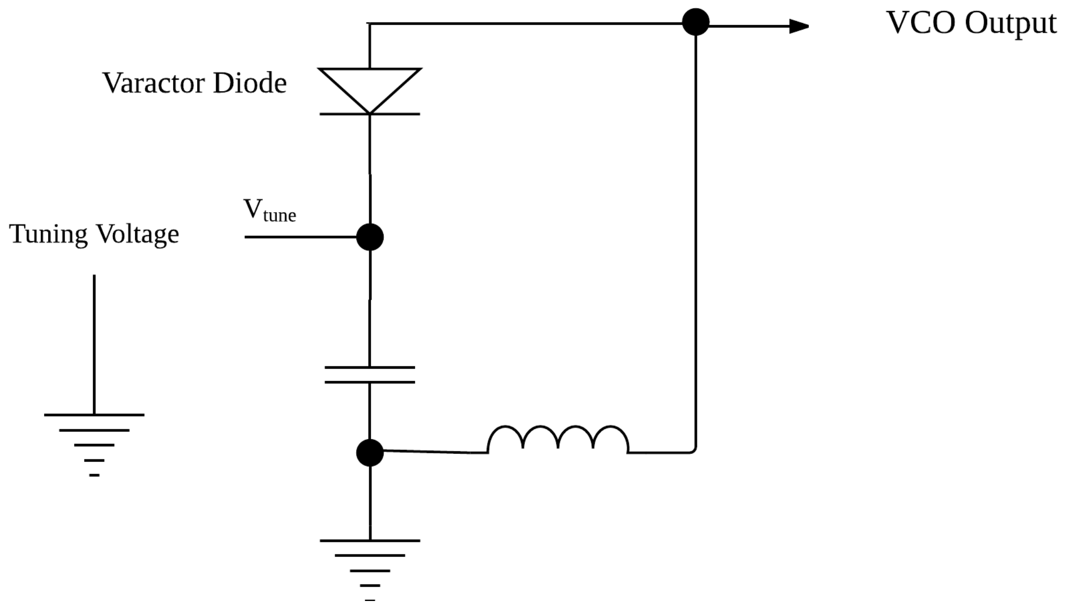


Figure 3.1: Basic design of a VCO.

a spread of frequencies about the spectral line. [12, p. 518]

Fig 3.1 shows an illustrative block diagram of a VCO. The capacitor and the inductor together determine the base oscillation frequency. The tuning voltage is supplied to the diode in the diagram. The diode is a varactor diode that varies its capacitance across the junction when the voltage across auction is changed. Consequently, the output frequency is controlled by the tuning voltage (applied across the varactor diode).

The VCO we use is MAOC-009269. From Fig 3.2 we can see the output frequency is increasingly nonlinear with increasing tuning voltage, but the output frequencies under different temperatures remain fairly close, not to mention we will make sure the temperature of the device stabilizes before the experiment. [1]

We may also find the output power's dependence on tuning voltage. Cheyenne

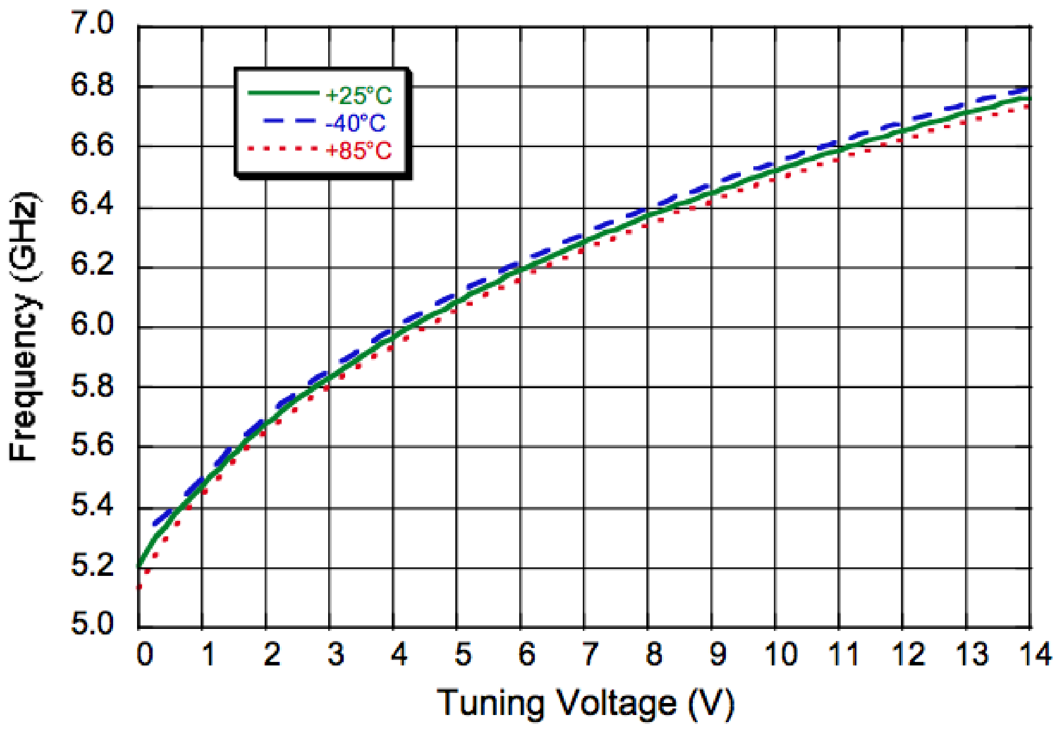


Figure 3.2: Output frequency's dependence on tuning voltage. Tuning voltage is the voltage directly across the varactor diode, not to be confused with supply voltage, which is the stable DC supply to power the VCO board. [1]

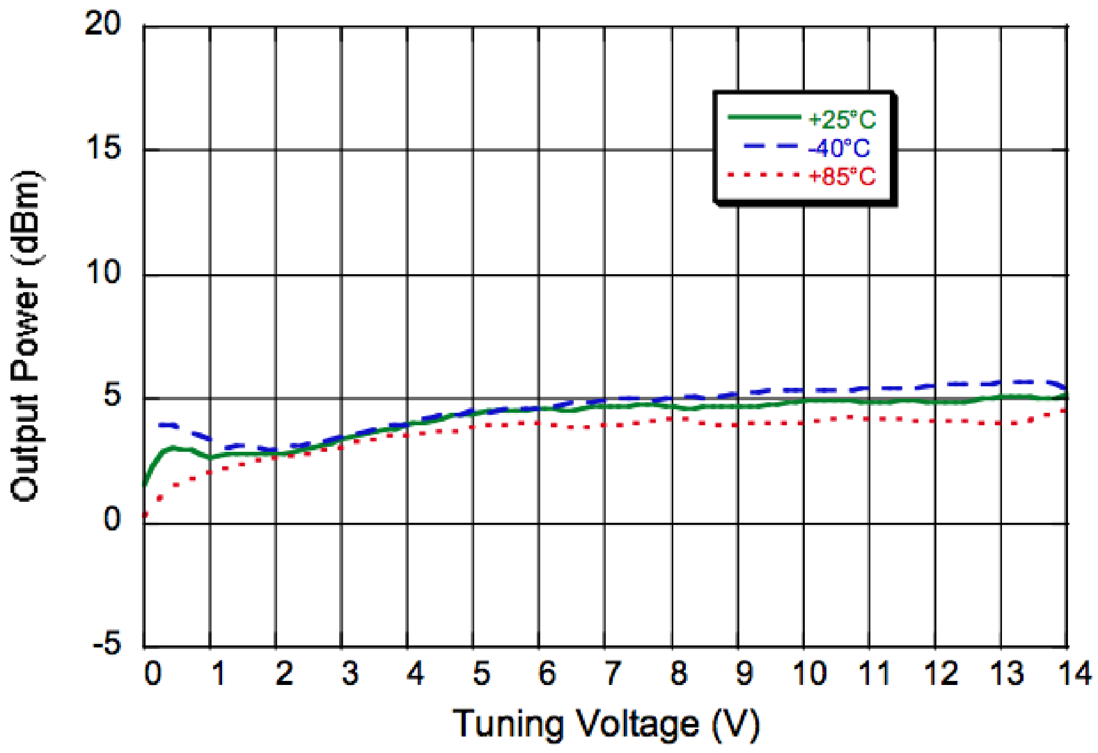


Figure 3.3: Output power's dependence on tuning voltage. [1]

Teng [7] determined a working power of 20 dBm. Without an option to change the amplify the output signal on our board, we have to carefully apply amplification and attenuation to reach the exact power level. This procedure will be presented in a later section.

In the next section, we will present the phase-locked loop, a control system that locks both frequency and phase of the output signal to the clock signal, further improving the stability of our modulation signal.

3.1.2 Phase-Locked Loop

A phase-locked loop (PLL) is a control circuit that locks a signal to another one, so that the two will have the same phase and frequency. By implementing the PLL, we can lock the VCO output to a reference signal, so that the stability of the reference signal can be transferred to the VCO output, considering any fluctuations in the tuning voltage introduce error to the output frequency.

A PLL consists of a phase detector, a loop filter (low-pass filter), a frequency source, a feedback divider, and a reference divider. A portion of the signal generated by the frequency source, in our case the VCO, goes through the feedback divider and gets its frequency divided by a set value N. Meanwhile, the reference signal is fed to the reference divider, and also reduce its frequency to $1/R$ of the original level. The values of N and R are determined by the feedback divider and the reference divider respectively such that

$$\frac{f_{ref}}{R} = \frac{f_{VCO}}{N} \tag{3.1}$$

This way we are making sure we are compare their frequencies at the same level. The necessity to implement the reference divider arises when we require a certain spacing in the output signal, and that spacing is uniquely determined by the divided-down reference signal whose frequency is $\frac{f_{ref}}{R}$. [13, p. 134-144]

Both divided signals are loaded into the phase detector to match their phases and frequencies. If both match up, the phase detector will generate no output any thus the whole loop is "locked." However, if the VCO frequency (divided down) or phase drifts from the reference signal, the phase detector will generate a negative feedback

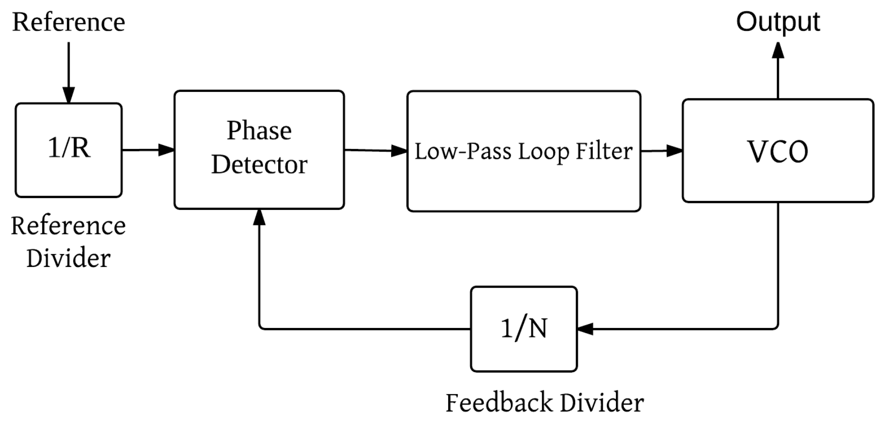


Figure 3.4: Block diagram of a typical phase-locked loop. This type of PLL is also called a integer-N PLL given that the feedback divider divides the VCO output by an integer, N.

signal that is combined with the VCO output to correct the errors. For instance, if the phase of the VCO output shifts, the phase detector responds by modifying the control voltage so that the VCO output catches up with the reference. Similar for frequency correction, the feedback signal addresses the difference between the two inputs, and modifies the VCO output by superimposing an negative error to it. It is possible to increase the resolution of the output signal by replacing the integer divider by a fractional divider, thus allowing the reference signal to be less reduced. This type of loop is called fractional-N PLL, but is not of our interest in this experiment.

We can see from Fig 3.4 a loop filter is installed between the phase detector and the VCO. This low-pass filter is set in place primarily due to the possibility that some of the reference signal power may be delivered to the output and cause a spurious level

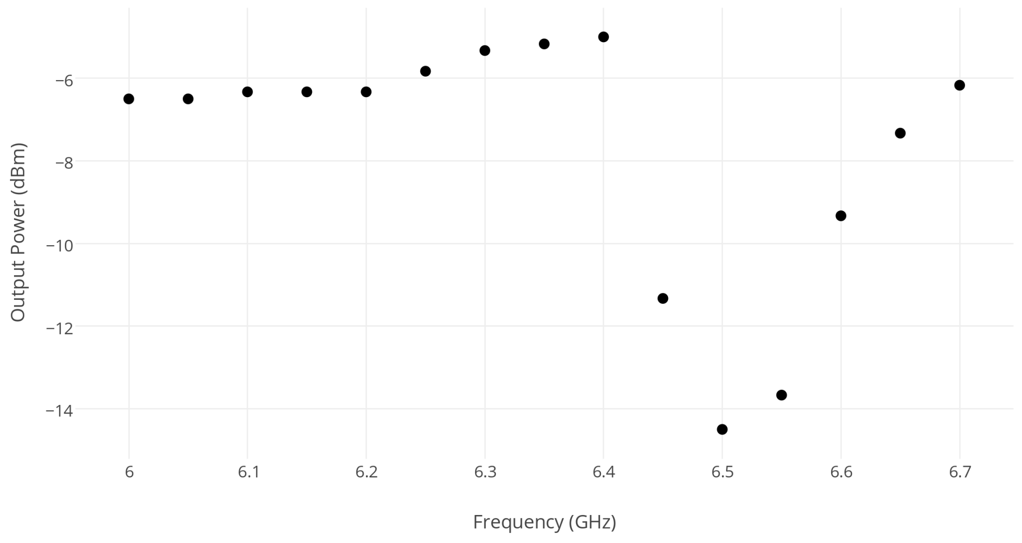


Figure 3.5: We measure the actual output power levels at different frequencies. The result differs from Fig 3.2 and Fig 3.3. Although the actual cause is unknown, but it is unlikely the VCO solely would perform so much differently from specified, so we suspect that the evaluation board attenuates the output at high frequency range.

in the output. Referred to as reference spurs, these undesired offsets in output come from instrumental imperfections in the phase detector. Since the spurs come directly from the reference signal, their frequencies are high compared to the feedback signal, therefore can be eliminated by a low-pass filter set to allow only feedback signal to pass through. Other considerations that motivate loop filter installation may include the ability to measure the overall stability of the loop, as we may alter the filter setting to find the ideal range in which the loop can achieve locking.

The PLL we use is Analog Devices ADF4108. It tunes over an 8.0GHz bandwidth that suits our interest. The PLL is integrated on an evaluation board, UG-160, that allows computer programmability. The board takes in a reference signal up to

250MHz in frequency, but we use 10MHz, the frequency of the lab master clock. To synchronize all the devices within the lab, we will ultimately use the lab master clock on our PLL in the long run, but in this initial test we will use a 10MHz reference output of the spectrum analyzer that we use to measure the output frequency and power. This makes sure the PLL and the analyzer is clocked by the same source. The result is a sharp peak at 6.6GHz on the frequency spectrum, as seen in Fig 3.6. From Fig 3.5 we can see the output power's dependence on frequency. There is huge discrepancy between the oscillator's documentation and the actual performance, and this may be due to the attenuations from the evaluation board, the wires we use, or the spectrum analyzer. [14]

We successfully obtain a signal at the target frequency. We need to implement amplifiers and attenuators that work at this high frequency range. We can use fixed-gain (or loss) type or variable-gain type device for each type of modulation, but considering the simplicity of our goal, fixed-gain amplifiers and fixed-loss attenuators would serve us just right. The next section will discuss how do we choose amplifiers and attenuators to reach accurate power level.

3.1.3 Amplification

Our goal is to increase the output power by 29 dB. It is hard to find a single amplifier that has this amplification level. We will use two amplifiers to go beyond 20 dBm first, then apply attenuators to bring the power level down, given that we have abundant choices of attenuators.

When we choose our amplifier, we have to look at not only the frequency range and gain, but also the maximum output power. Before damaging the device, the

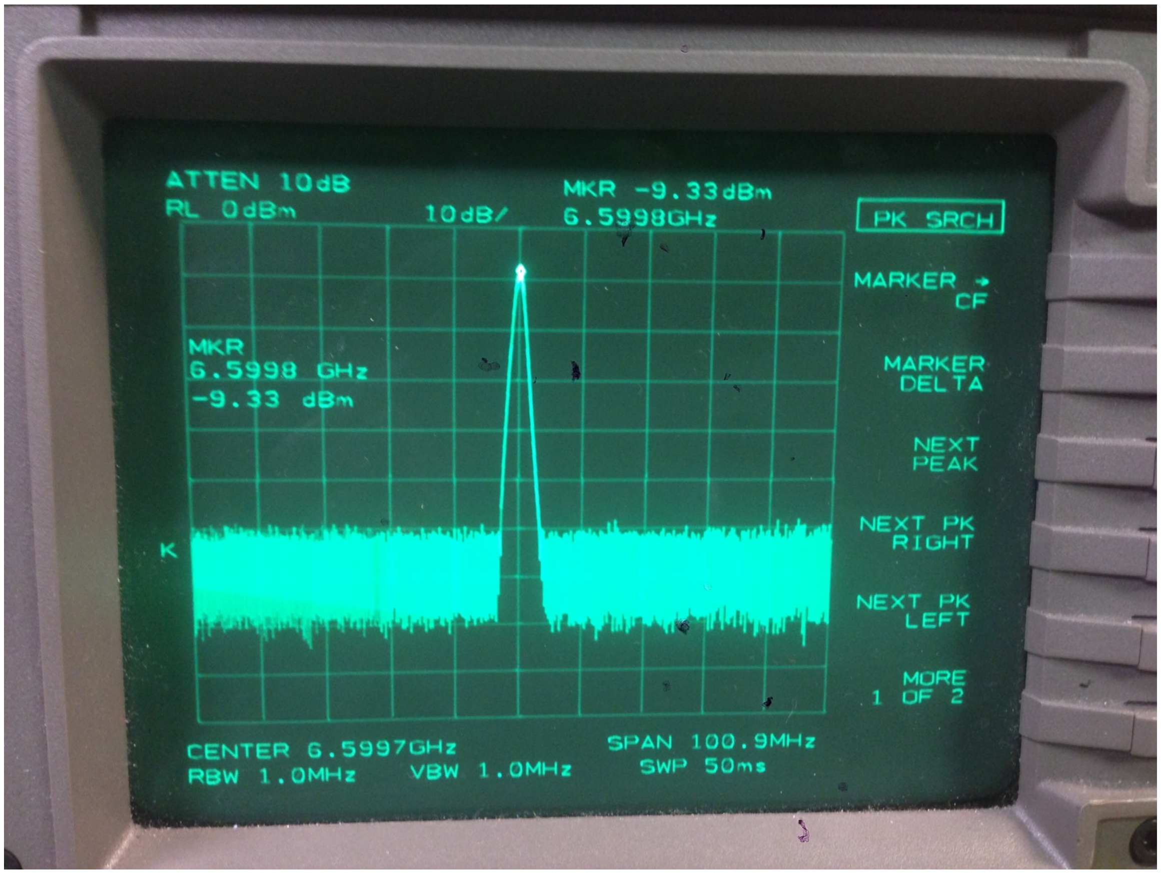


Figure 3.6: Spectrum analyzer displaying a sharp peak at 6.6GHz on the frequency spectrum of the output signal. The output power is -9.33 dBm at this point.

output usually saturates at a certain level due to the increasing nonlinearity. We characterize this behavior as the compression region. Ideal amplifiers guarantee a linear relationship between the input power and the output power, but practical amplifiers exhibit nonlinearity, and the difference between the theoretical response curve and the actual response curve becomes larger with larger input level. The point where the theoretical value is 1 dB higher than the actual value is called the 1 dB compression point, marking the boundary where nonlinearity becomes dominant. The actual response curve usually becomes flat very quickly after the this point, so we can regard the output power level at the 1 dB compression point as the maximum output power of the amplifier.

The first amplifier we use is Mini-Circuits ZX60-183+. It works from 6GHz to 18GHz, and has a typical gain of 24 dB. The 1 dB compression point is at 18.2 dBm when the signal frequency is 6.6GHz, meaning that we cannot go beyond this level with this single amplifier. [15] We then have to install another amplifier, Mini-Circuits ZRON-8G. This amplifier works at range 2-8GHz and has fixed gain of 20 dB, and 1 dB compression point at 20 dBm. We can refer to Fig 3.8 to find that the amplifier should achieve a gain of more than 22 dB at 6.6GHz, but would saturate before that due to nonlinearity of the response. The ZRON-8G amplifier can take 10 dBm at maximum as its input, so the signal must be attenuated before it. [2]

We start testing the output power by connecting the two amplifiers together, and applying different attenuators in between.

We see from the table for attenuation vs output power that the ZRON-8G amplifier starts to saturate before 20 dBm. Although the device goes to 19.67 dBm after it starts to saturate at 19.50 dBm, we should regard 19.50 dBm as our maximum possible

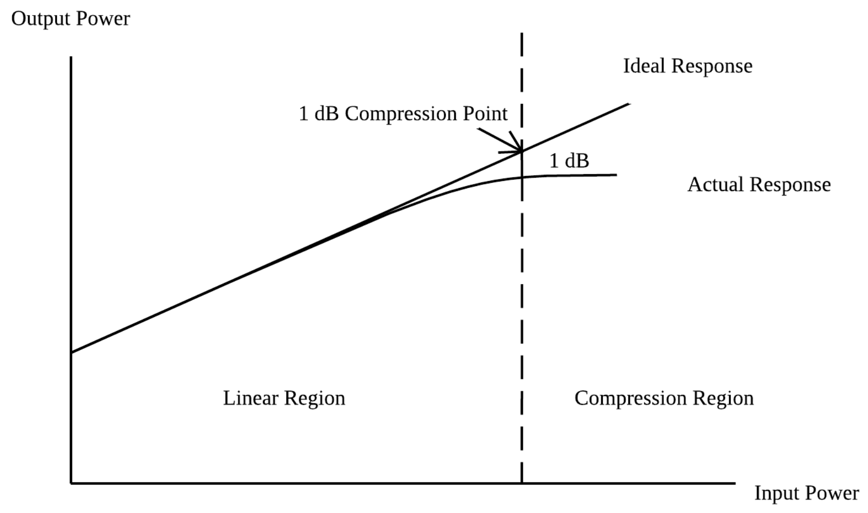


Figure 3.7: The region left of the 1 dB compression point is the linear region, and the region right of the compression point is the compression region. Practically we can take the output power level at the compression point as the maximum power.

Attenuation	Output Power (dBm)
10 dB	18.50
9 dB	19.00
8 dB	19.33
7 dB	19.50
6 dB	19.50
5 dB	19.50
4 dB	19.50
3 dB	19.67
2 dB	19.67
1 dB	19.67
0	19.67

Table 3.1: Measured output power with different attenuators connected between the amplifiers.

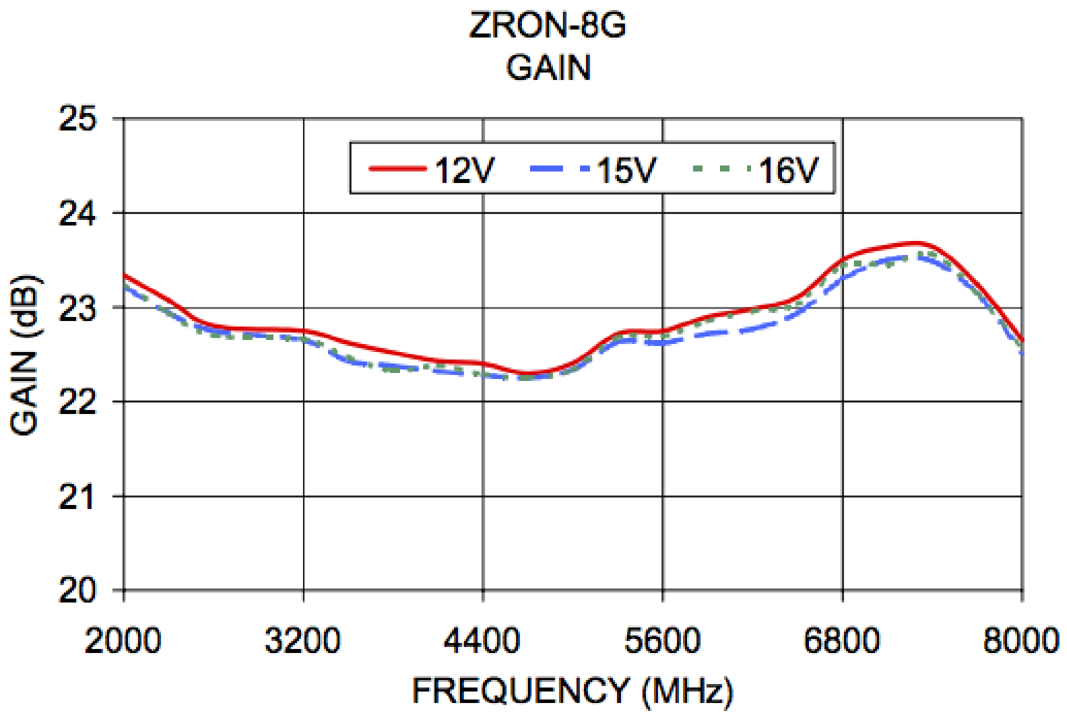


Figure 3.8: The typical gains of the ZRON-8G amplifier at different frequency levels. [2]

level, since after this point we are inside the compression region, and any increase in input power could bring potential instrumental damage. The specification sheet of the amplifier also states that input power cannot surpass the 1 dB compression point to avoid overload. [2]

The highest power we can get from our system is therefore 19.50 dBm. We expect this to be a working power. Experimental verification is still necessary. If it turns out we need slightly more power for the detuning signal, a third amplifier may be needed to be installed in serial, or replace the ZRON-8G amplifier. For now, we can conclude this section about the detuning system, and move on to the pulse sequencing system.

3.2 Pulse Sequencing System

In both cooling schemes we will use to cool the ion, our laser beam has to be modulated to different frequencies. In addition, the laser beam needs to shine on the ion for precise durations, especially in resolved sideband cooling where π-pulses make the population move from one state to another completely. This time we will be working in the very high frequency (VHF) range in the RF spectrum. To avoid ion thermalization between pulses, the pauses between pulses have to be in the scale of microseconds, given that ions thermalize in milliseconds. A VCO is no longer the ideal choice for signal generation since it is not suitable for fast pulse sequence generation. Instead, we will present another RF synthesis instrument called a direct digital synthesizer (DDS) that has certain advantageous feathers in pulse sequencing, making it a more favorable synthesis solution than the VCO and PLL combination.

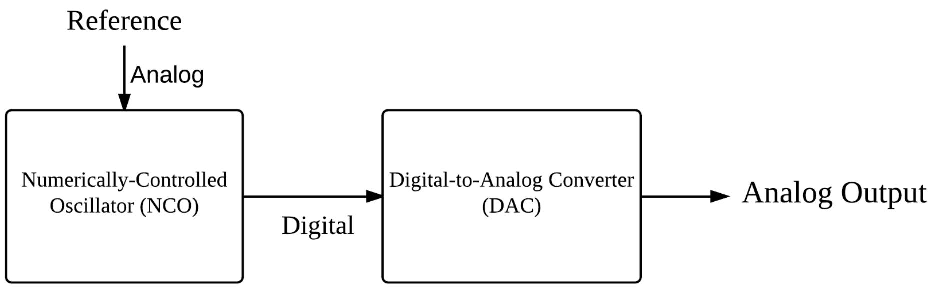

Reference
Analog
Numerically-Controlled Oscillator (NCO)
Digital
Digital-to-Analog Converter (DAC)
Analog Output

Figure 3.9: Fundamental design of the direct digital synthesizer.

3.2.1 Direct Digital Synthesizer

Like a VCO, a DDS also takes in a reference signal and generates a waveform, but the basic mechanism behind the two types of synthesizers are very much different. Inside a DDS, a numerically-controlled oscillator (NCO) receives the reference signal, then creates a numeric representation of a waveform, essentially a digital signal that contains all the waveform information. Functionally, the NCO is a look-up table for the waveform and a counter. The digital output of the NCO is fed to a digital-to-analog converter (DAC) to acquire an analog output signal. For the purpose of this thesis, it is important to know how to program the DDS so that the NCO produces the representation of the desired waveforms.

Compared to VCO, DDS has a number of advantages in pulse sequencing, with the most noticeable being the ability to change the frequency of the output. The complete waveform information is stored in the memory of the NCO, and the NCO updates its output at each clock cycle from a 1GHz clock. Notice the since the clock frequency determines how fast we can update our output, the DDS excels in pulse sequencing since it is using a 1GHz clock, much faster than the 10MHz clock the VCO uses. With such splendid frequency agility, we can achieve arbitrary frequency modulations within microseconds. Another feature of the DDS is its relatively low phase noise. In an ordinary PLL, the feedback divider in the loop can reversely take in the reference signal, amplify the phase noise of the reference, and let the phase error bypass the phase detector. We can reduce this type of noise introduction with DDS since the analog output is derived by dividing the reference signal instead of multiplying it.

The DDS also excels the VCO in frequency resolution. The frequency resolution of a VCO inside a PLL is limited by the reference signal, especially if the PLL is integer-N. The DDS does not escape from the reference limitation, but since it uses reference signal at much higher frequency, the PLL resolution limitation is automatically eliminated. This perk helps DDS prevail in pulse sequencing for sideband cooling as the effectiveness of the cooling scheme depends on how closely we can modulate the laser beam to desired frequencies. [16]

It is worth noting that a typical DDS cannot work in high frequency ranges like a VCO. Certain circuit designs allow the DDS to surpass this limitation, but our experiment does not impose such necessity. Our diode laser and the detuning system will position us close to the frequency range where Doppler cooling and resolved sideband cooling work. After that we only need modulation signals in the scale of 10's to 100's of MHz.

The DDS we work with is Analog Devices AD9910. The synthesizer generates up to 400MHz analog output, and has frequency resolution at least better than 0.23

Hz. AD9910 features 8 registers that can store 8 waveform profiles. Each profile is a unique combination of frequency, amplitude, and phase. It integrates a 14-bit digital-to-analog converter with sampling rate 1 GSPS (giga samples per second). [16] Later we will discuss how to program waveform data into the DDS profiles and switch between profiles. This procedure is what we want to use to generate arbitrary pulse sequences. For now, we are missing a building block in the system structure. Our DDS can store waveform data, but not sequence information. We cannot use a computer to perform simultaneous control since computers tend to have irreproducible executions, given that they are processes running in the background on a computer that could slow down or even pause the execution. An instrument that can achieve fast multi-channel control and synchronize its clock with the DDS is needed. This is where the field-programmable gate array (FPGA) comes into play.

3.2.2 Field-Programmable Gate Array

A FPGA is an integrated logic circuit consisting of logic components programmable by users, giving rise to ”field-programmable” in its name. It is capable of performing complex logical tasks, and can be programmed using a computer. In our experiment, the idea is to turn an FPGA to a central control unit for multiple DDSes so that a programmed FPGA can solely instruct the DDSes to generate the desired pulse sequences without subsequent commands from the computer. [17]

The FPGA+DDS solution for our experiment is the Milldown DDS board manufactured by Enterpoint. It consists of 4 on-board AD9910 DDSes and a Xilinx Spartan-6 FPGA. [18] Since each channel has 8 available profiles, a single board can grant 32 output modes. See Fig 3.11 for an actual picture of the board. The board

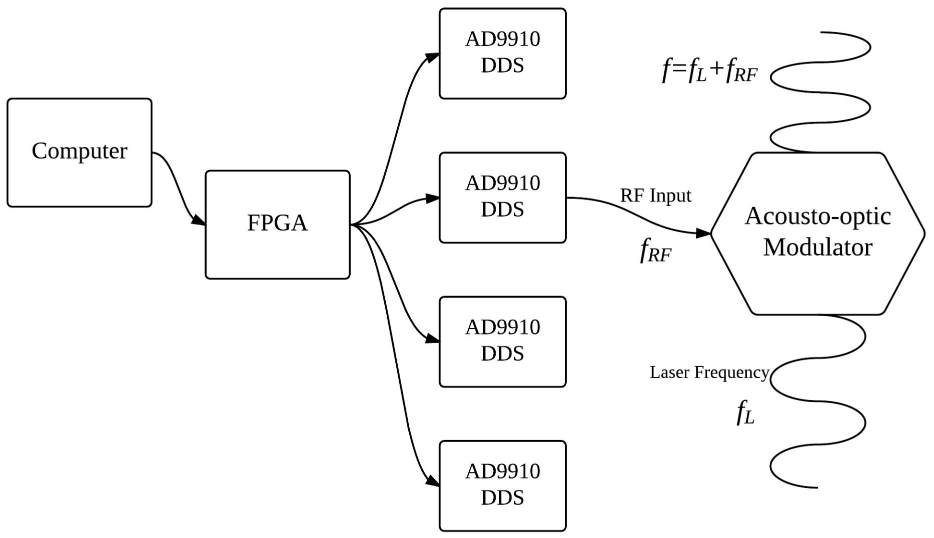


Figure 3.10: With a single FPGA, we can command multiple DDSes simultaneously to perform synthesis tasks. The diagram shows the output signal from one DDS being used to modulated the laser beam frequency through an acousto-optic modulator.

features 4 variable-gain amplifiers (VGA) to achieve amplitude modulation when used in conjunction with a DAC (integrated in the DDS).

One convenient feature of this integrated system is its ability to synchronize the FPGA and the 4 DDSes by a 1GHz clock signal. There is an on-board clock synthesizer, but we will still use external reference for stabilization and synchronization purposes. See Fig 3.12, Fig 3.13, and Fig 3.14 to find how a single 1GHz reference is used to clock the 4 DDSes on a board.

Milldown's design lets the DDSes be clocked by the external reference first. The board also clocks the FPGA using the same reference signal, so that phase jittering in the output is minimized since all the devices are synchronized. We can see how the FPGA is clocked from same source as the 4 DDSes from Fig 3.14.

To extend the number of channels, we will use more than one board at the same

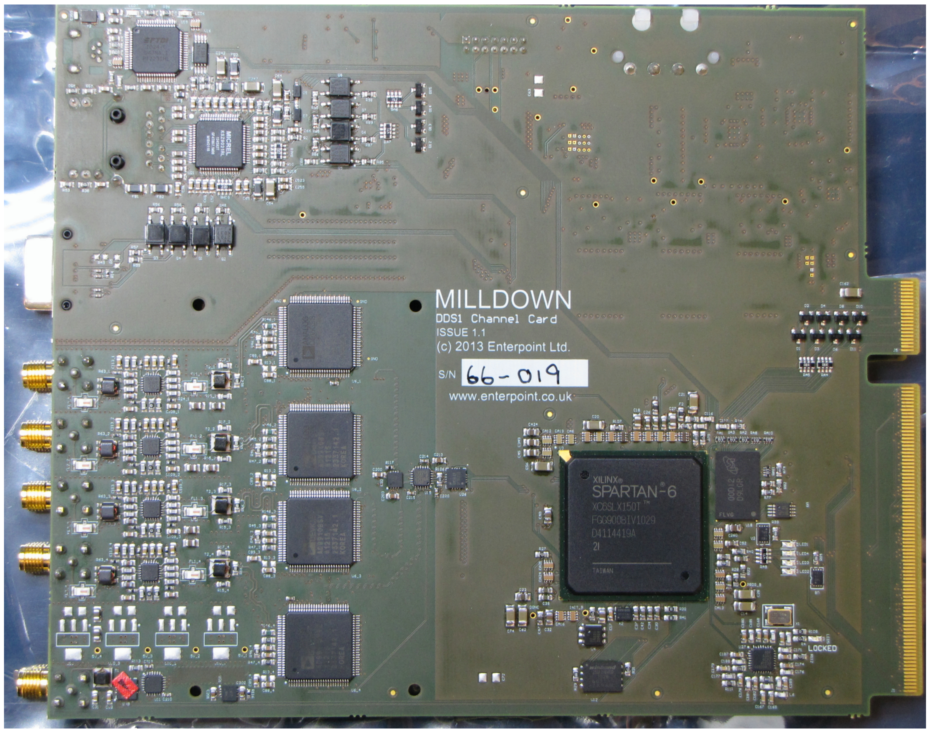


Figure 3.11: Front view of the Milldown DDS board. The 4 medium-size chips on the left are the integrated AD9910 synthesizers. The large chip on the right is the Spartan-6 FPGA. The top 4 SMA connectors on the left are DDS output ports, while the bottom connector is used as reference input. The region on the top-left is the internet-control unit, which we will not use in this experiment.

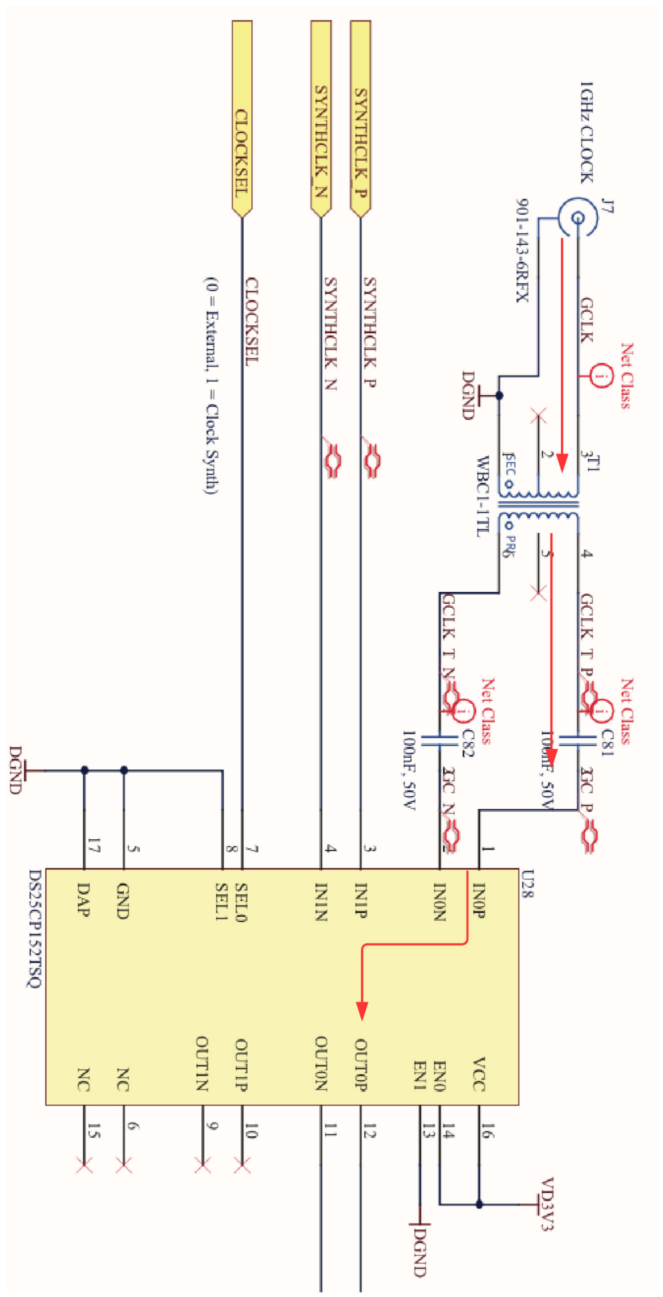


Figure 3.12: The 1GHz reference signal enters through the J7 connector, and goes to the DS25CP152TSQ crosspoint switch as one of the selectable inputs. The other input is STNTH_CLK, the reference signal from an on-board clock synthesizer. CLOCK_SEL is the selection signal provided by the FPGA. We do not use on-board clock in our synthesis, so we should use 0 on CLOCK_SEL. Actually, the FPGA automatically overrides the on-board reference if it detects external reference input. The external signal is then sent through the output port. [18]

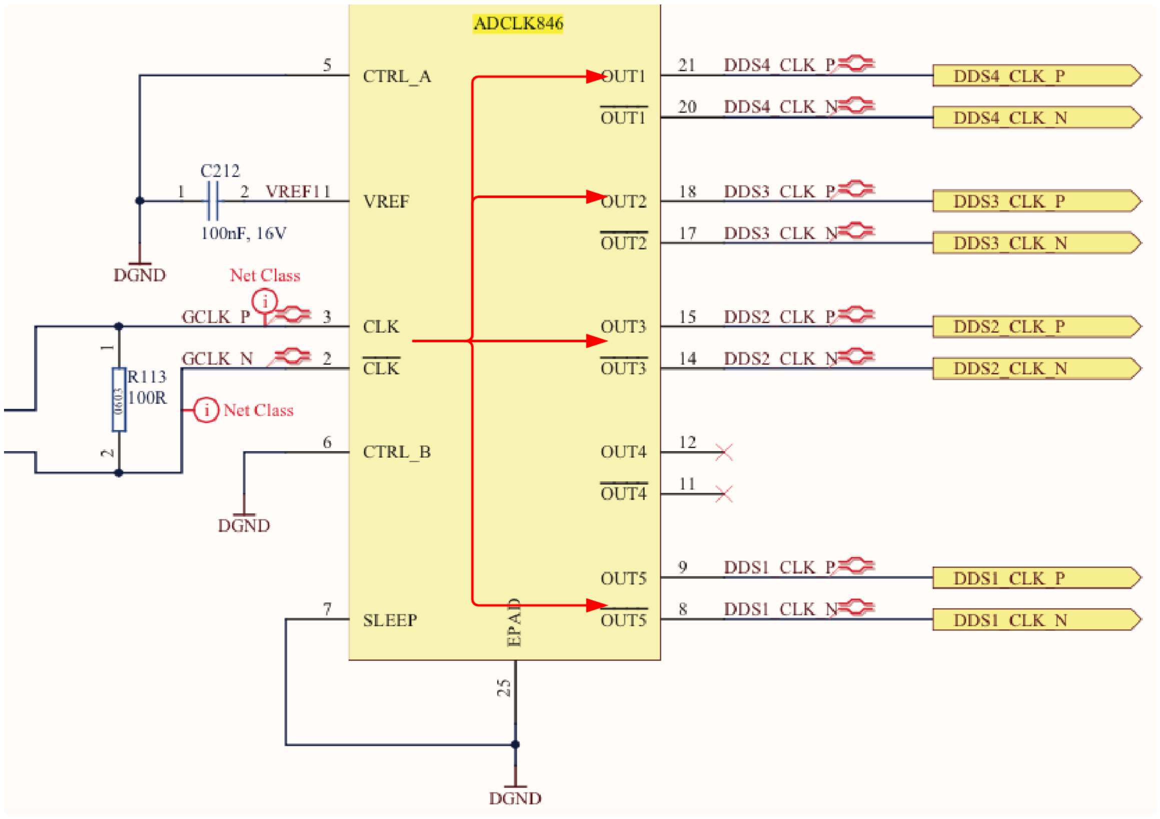


Figure 3.13: Following the the clock switch in Fig 3.12, the external reference signal acts as the input of an ADCLK846 fan-out buffer. The fan-out buffer is a clock multipliers that creates copies of an input clock at the same frequency and feeds them to multiple devices. We see that the clock signal is multiplied into 5 outputs and then clocks 4 DDSes (one output is terminated). [18]

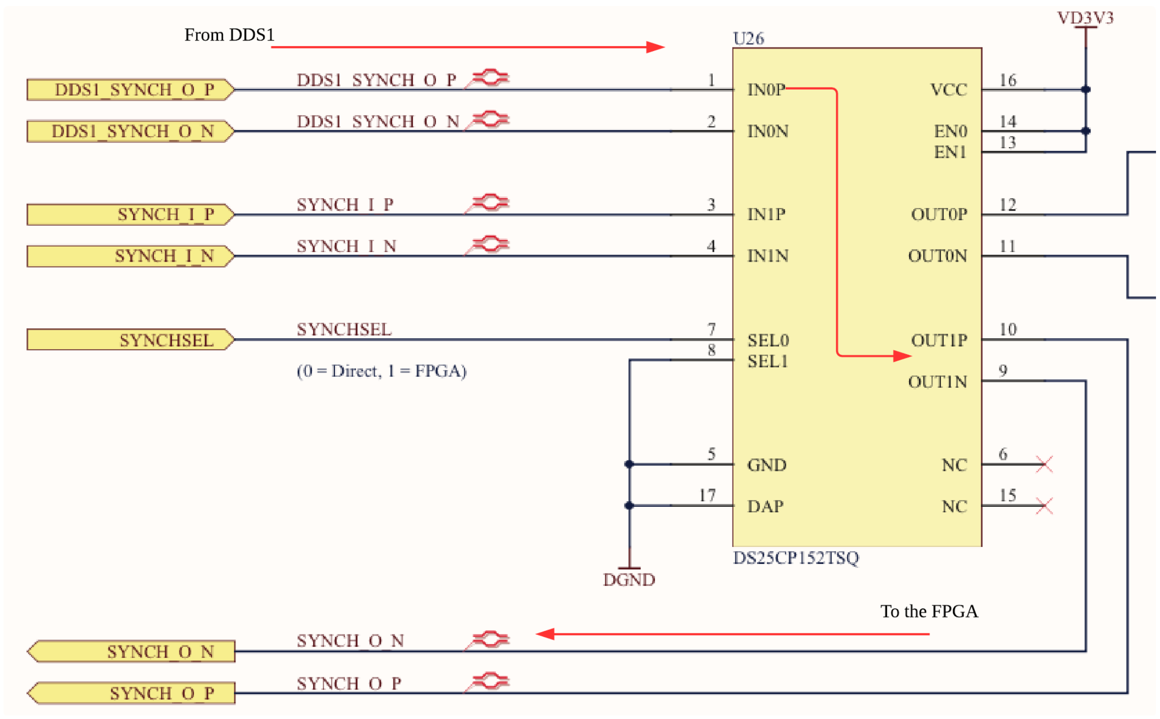


Figure 3.14: The external reference is used to clock the 4 DDSes. DDS also has a synchronization output that has the same frequency and phase as the reference signal, and this signal is used to clock the FPGA through SYNCH_O port after going through another DS25CP152TSQ switch. This way, the 4 DDSes and the FPGA are clocked by the same external source. [18]

time. Milldown has a backplane product that supports up to 8 DDS boards via standard PCI-E connectors. It also uses a single power input to power all the boards, and has an optional feature to route a reference signal to the 8 boards. [19] We will install an 8-way splitter to distribute the clock signal for simplicity, and this will make sure every single major device, DDS or FPGA, is clocked by the same external clock signal. See Fig 3.15 for an actual picture of the backplane.

Unlike our PLL system, the Milldown board does not come with readily available software that allows simple programmability. Hence we have to find a way to configure the FPGA and the DDSes through USB ports. In the next section, we will present the process of configuring the system using computer languages.

3.3 System Control

Having derived the pulse sequences required for resolved sideband cooling, we wish to translate the waveform information to commands to be executed by the FPGA. Each pulse in the sequence may have different frequency, amplitude, or phase, and each of these signal control parameters has to be pre-defined before any output. Each frequency+amplitude+phase combination will be stored as a waveform profile in one of the AD9910's 8 profile registers to be selected later on.

We use a Python script provided by Vlad Negnevitsky to define the signal control parameters, and load them into the profiles. This script has to be used in conjunction with a FPGA firmware, also developed by Vlad Negnevitsky, so that the FPGA can understand Python commands and turn them into digital signals as instructions for the DDSes. We will now introduce the process of defining control parameters step by step.

The idea is to define the waveform in any sufficient way, so simplicity is preferable. For each waveform, we define frequency in MHz, amplitude in percentage, VGA amplitude in percentage, phase in degrees, and profile number from 0-7, then group them into a text file. From here, the commands need to separate ways– frequency, amplitude, and phase modulation happen inside the DDS, so these two parameters shall be directed to the DDS via the FPGA, while the VGA amplitude command goes directly to the VGA.

In Vlad Negnevitsky's script, each DDS waveform profile is a 24-bit binary chain containing frequency and phase information translated from our simple definition. When we execute it, the waveform information, also called payload, gets sent to the FPGA. The instruction that sets the VGA also routes to the FPGA, and when it arrives, it tells the FPGA to send the DDS payload instruction to the DDS, thus filling a profile. One thing to keep in mind is that the every instruction we send to the FPGA will trigger the FPGA to send an I/O update signal, and this is the command that initializes and updates the DDS output. For instance, if we keep programming the DDS profiles, the DDS will always generate a signal based on the last profile loaded into it, because of the I/O update signal attached to the VGA instruction. The importance of the I/O update signal does reveal itself at this stage, but later on when we perform profile switching in section 4.1 the I/O update signal becomes a very important factor in determining the pulse sequence.

The workflow of the script is as follows: First, the program loads the waveform settings stored in the text file, then translates them into binary commands. The Python script then scans the computer serial ports and locates the port of the board.

Next, it initializes the DDSes by clearing out all the profile registers on the DDSes, namely writing them with all zeros. The last step is the actual writing: the program sends the instructions to the FPGA as described above. It is worth noting that with an in-program timer, we measure the run time of the entire script is more than two seconds, meaning that the whole process of writing profile registers on the four DDSes and also the VGA settings takes about that much time to complete. Of course, we would not want to update the output by writing a profile and sending an I/O update signal, simply because this takes too much. The DDS actually supports simultaneous direct control on output via a high speed parallel data input port, but we will present a more agile method to achieve arbitrary switching between the eight profiles on a DDS by implementing a parallel input of digital triggers in the next section.

3.3.1 Transistor-Transistor Logic Triggers

When shaping up a pulse sequence, a pulse's frequency, amplitude, phase, and duration often need to be different from the preceding pulse. By grouping these 4 parameters into a set for a particular pulse, we can create a sequence by switching between these sets with right timing. In the previous section, we introduced how to program frequency, amplitude, and phase information into the DDS's profile registers. The remaining questions are– how do we set the duration of each pulse? And how, for each pulse, do we tell the DDS which profile to base its output on? Keeping these requirements in mind, we ought to find some external signal source that can synchronize with the DDS output and also carry the information for the choice of profile. The DDS itself contains three input channels that allow profile selection by logic signals. Fig 3.17 shows the pins used for digital logic signal inputs.

We use transistor-transistor logic (TLL) signals as the control inputs. TTL is a logic gate digital circuit capable of generating digital logic signals. Such signal consists of sequence of pulses that contain logic information. A logic high, which is 3.3V in our case, means "1" while a logic low, 0V with respect to ground, represents "0." We intend to create parallel TTL inputs as triggers, directing the DDS to select profile accordingly and setting the duration of each DDS pulse. This means for each DDS we will need three TTL lines (channels) to ensure full switching capacity. When we implement this method, it is important to keep in mind that since one DDS has only eight output modes, even if at some point in the output sequence, such as between pulses, we want to have a range where there is no signal, it would still take a profile set to zero amplitude. When we set the logic levels, we should make sure each 3-bit command is equal in duration, meaning that the three TTL lines should correspond to each other in a way such that they remain their logic levels for the duration of a pulse, until we need to switch to another profile.

We will illustrate the profile selection using an example. Consider having the DDS profile 0-4 set as following:

DDS Profile 0: Frequency=0 Amplitude=0 Phase=0

DDS Profile 1: Frequency=$2f_0$ Amplitude=$2a$ Phase=0

DDS Profile 2: Frequency=f_0 Amplitude=a Phase=0

DDS Profile 3: Frequency=$2f_0$ Amplitude=a Phase=0

We can make a pulse sequence using these 4 profiles. See Fig 3.19 for the DDS output derived from these profile settings and a 3-line TTL pulse sequence.

We use an NI PXIe-6536 digital I/O board manufactured by National Instruments for TTL lines. The board supports 32 digital channels, which means it can drive 10 DDSes or more, considering we need at most 3 lines for a DDS. Control of the board requires the use of Labview, and we wrote a virtual instrument that can load a waveform file in the format of hierarchical waveform storage (HWS) and drive the board for TTL generation.

Before we can send the TTL signals to the DDSes, we need a slight revision of the FPGA firmware as the original version written by Vlad Negnevitsky supports only one profile pin on each DDS. Because there is no way we can send TTL signals directly to the DDS because the profile pins are connected with the FPGA, we have to use input ports on the Milldown board to route our signals. Fortunately, we have abundant choices– the Milldown board has two 10-pin headers that can take in TTL inputs. In addition, there is another header on the backplane that has the same functionality. These pins are customizable– you can use a pin as manual I/O update, or you can send TTL signal to a pin to power-down the DDSes, but here, we change the FPGA firmware such that the TTL signals we send to the pins will be directed towards the profile pins on the DDS.

From here, we are ready to generate arbitrary pulse sequences by programming the DDS profiles and controlling the DDS output by implementing TTL triggering. In the next chapter, we will test our system by evaluating DDS pulse sequence output on an oscilloscope.

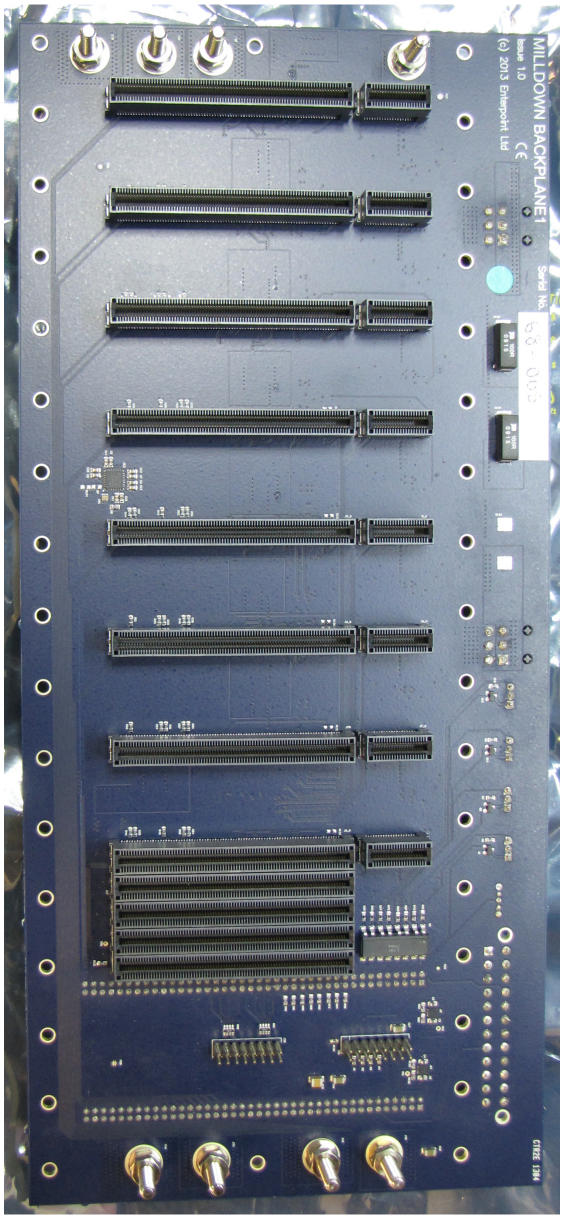


Figure 3.15: Front view of the Milldown backplane.

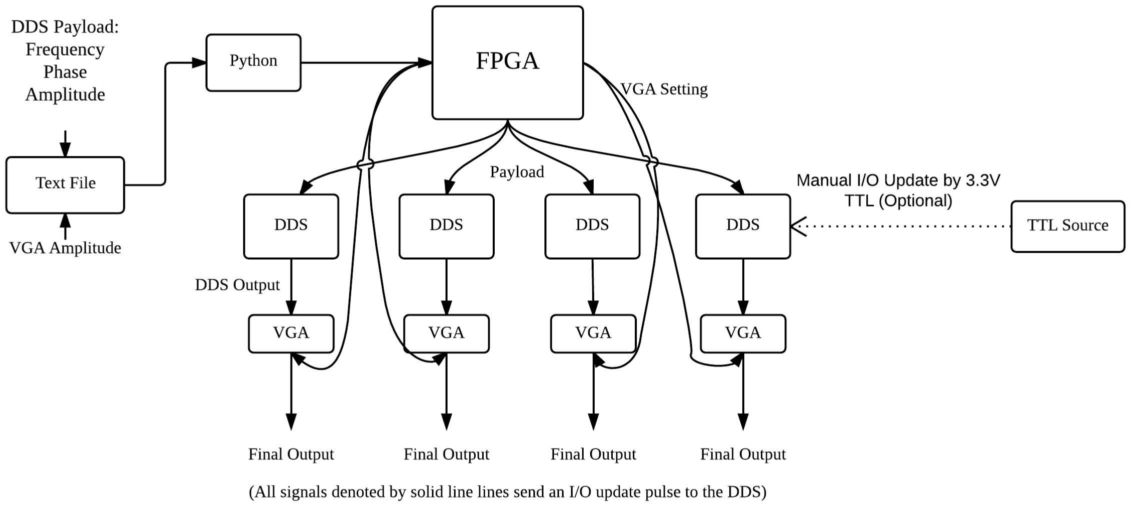


Figure 3.16: Workflow of profile programming. Notice that all the commands from the computer go to the FPGA first.

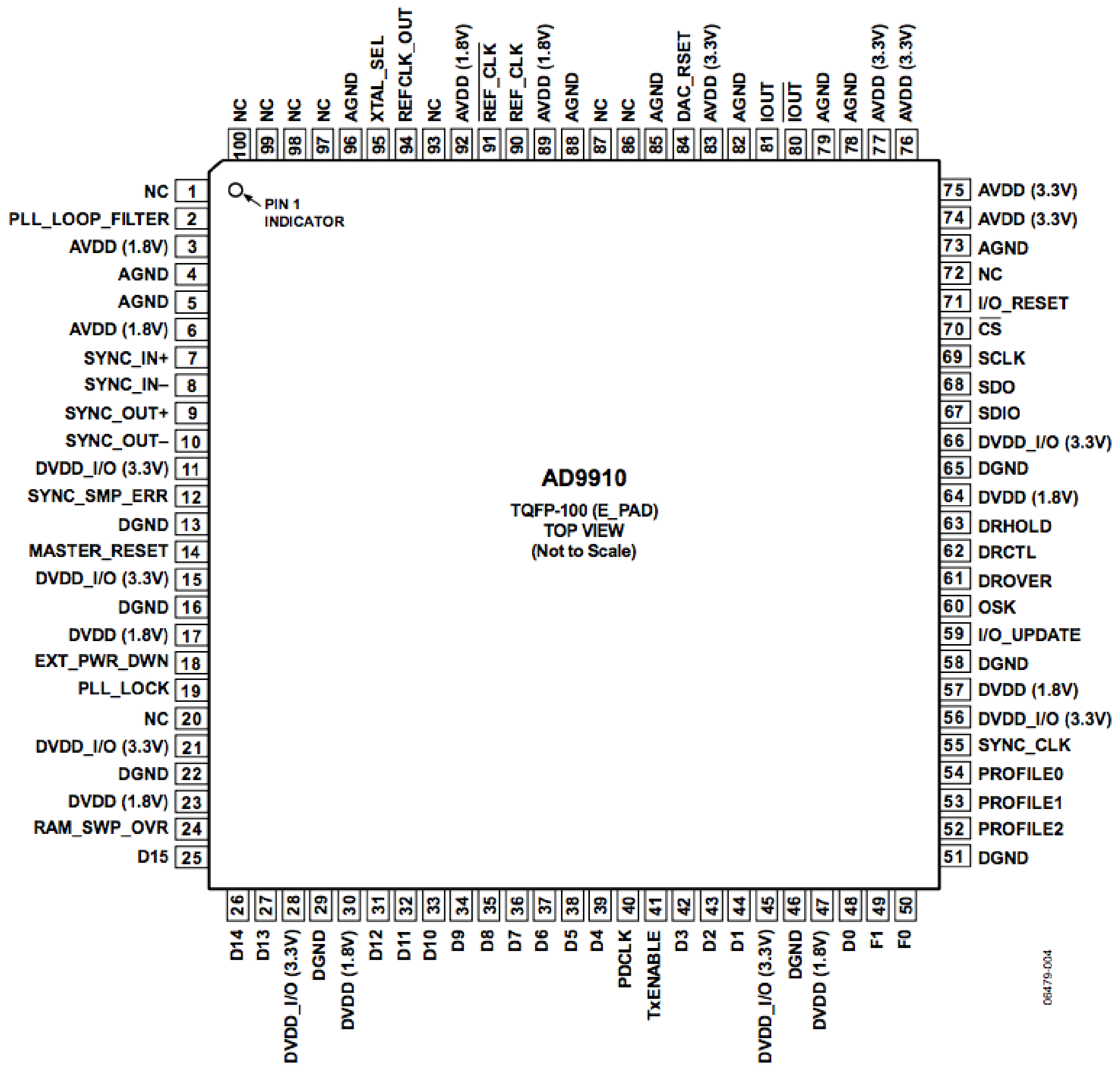


Figure 3.17: Pin layout for the AD9910 DDS. Pins 52-54 are used for parallel TTL inputs. Pin 54 constitutes bit 1 in the 3-bit command, while pin 53 and pin 52 are used for bit 2 and bit 3, respectively. [18]

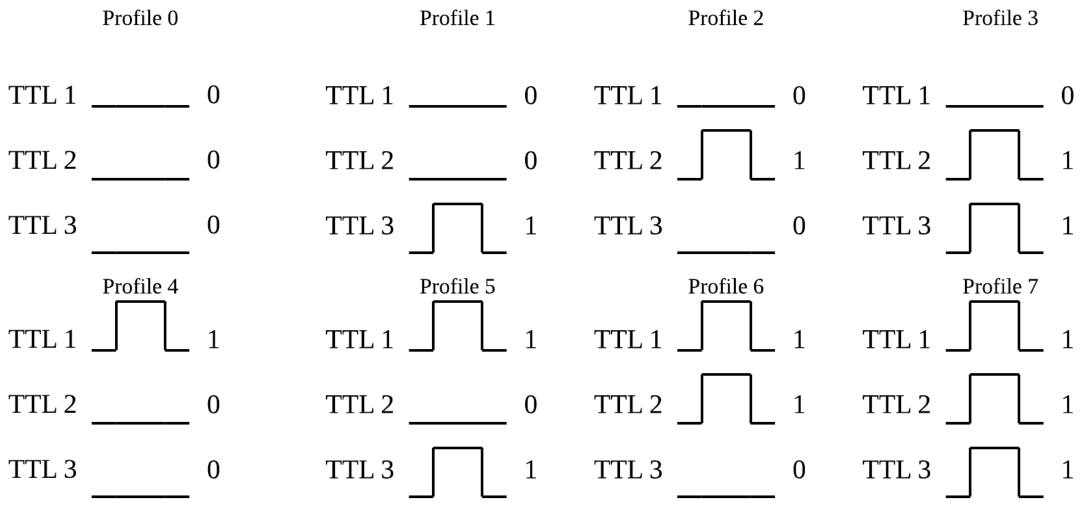


Figure 3.18: This figure shows how to map profile selection to a particular TTL combination. When showing a logic high or ”1” level, the figure shows a logic level change from low to high, then from high to low. for clarity. This may not be an actual process in practice, as the DDS perceives a logic low as a ”0” and selects profile accordingly. This means, if one TTL input is on the same logic level for two consecutive pulses, we should not change the logic level in between, unless we want to turn off the output for a moment.

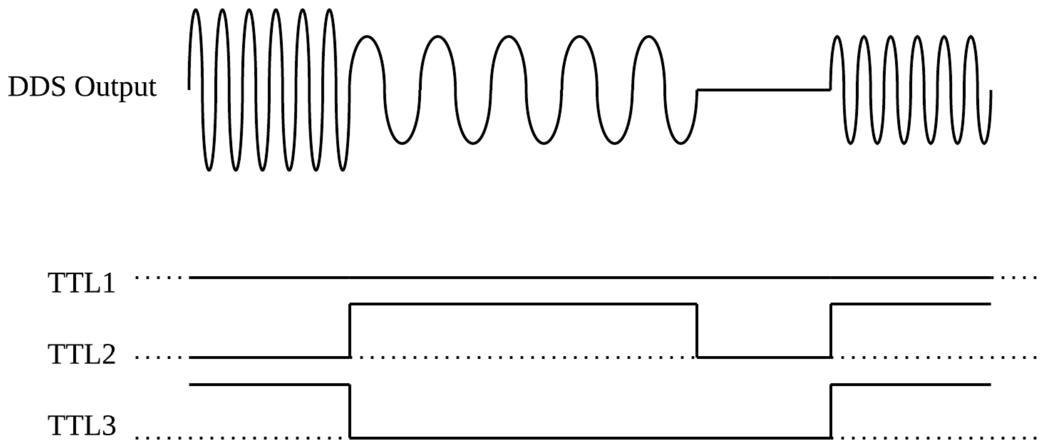


Figure 3.19: TTL lines triggering DDS output, creating a specific pulse sequence based on our commands. Dashed lines are logic low levels.

Chapter 4

System Performance and Discussion

To proceed with the cooling schemes, we need to test if our RF pulse sequence for laser modulation meets the requirements. We have so far constructed the pulse sequencing system by programming the FPGA and configuring computer-based control programs. We will now see if the system allows us to generate an arbitrary pulse sequence.

4.1 DDS Output Test

As presented in chapter 3, our DDS is capable of generating a signal based on the waveform we define, provided the payload does not exceed any of the DDS's maximum capacity, such as 400MHz in frequency. Before proceeding to pulse sequence creation, we first need to test the quality of single mode output signal. We will operate the DDS to generate a sinusoidal waveform based on our definition, and capture the actual output on an oscilloscope. Without any TTL inputs, the DDS operates on the

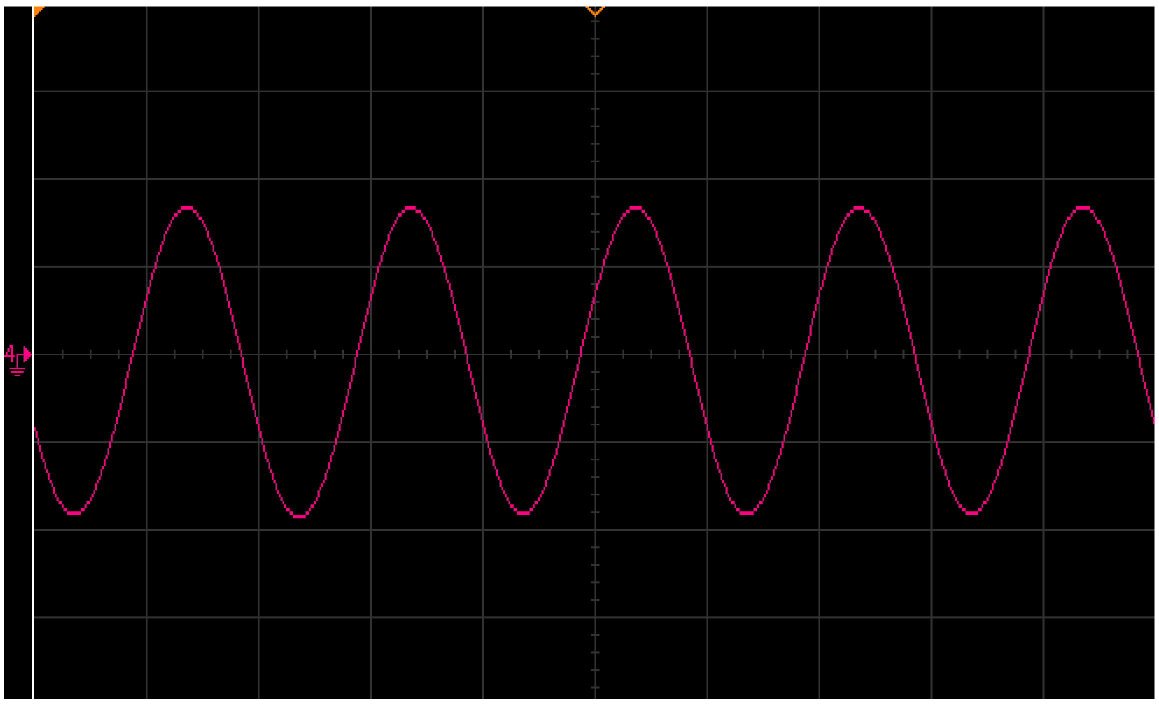


Figure 4.1: The sinusoidal DDS output based on an arbitrarily defined waveform. Each grid on the x-axis is five nanoseconds.

payload information stored in profile 0. So let us define a waveform in profile 0 as:

DDS1 Profile 0: Frequency=100MHz Amplitude=10 Phase=0

In section 3.3, we introduced how the DDS refreshes its output based on the I/O update signal sent by the FPGA. This means that we do not need any TTL triggers to update the output to this newly programmed waveform. We can simply run the Python script, then the DDS will commence the generation according to the first profile continuously. Fig 4.1 shows the actual output from the DDS. We can see it is a very stable signal without noticeable phase jitter– different cycles are clearly in phase with each other.

We mentioned in section 3.2.2 that the DDS board is integrated with an internal

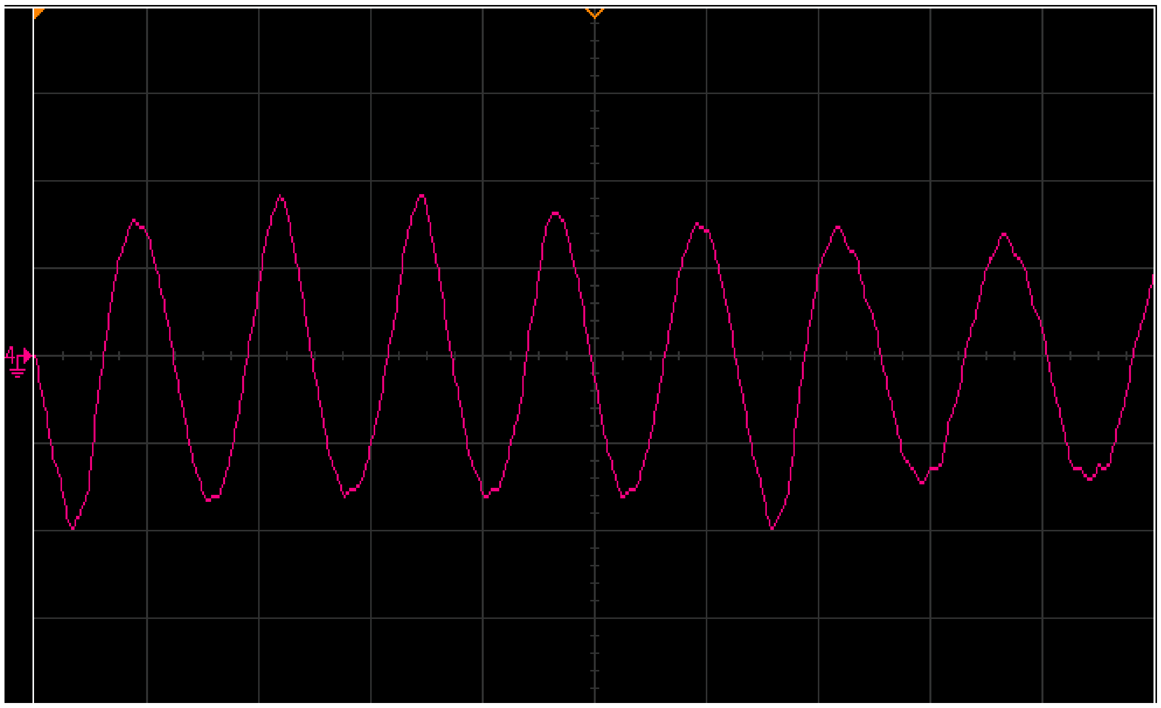


Figure 4.2: The DDS output when the internal clock is in use. We can hardly acquire anything useful in this signal, since the phase inconsistency is very observable even between two consecutive cycles. The amplitude of the wave fluctuates significantly as well. Each grid on the x-axis is 20 nanoseconds.

clock. For synchronization between the devices we use, we do not use the internal clock as the reference signal for the DDS. However, it is worth comparing the internal clock with the external clock for quality of the output. The FPGA firmware is programmed such that the external reference, if present, overrides the internal clock. We can simply disconnect the external clock to start using the internal clock. The result is shown in Fig 4.2. We can see the output has not only significant phase jittering, but also inconsistent shaping that can hardly be considered sinusoidal. This means the external clock excels in both device synchronization capacity and signal quality.

4.2 Profile Switching Test

We see that the DDS, when operating in single mode, can generate stable sine waves based on our definition of the waveform. We now ought to make the DDS work in multi-mode and create a pulse sequence using profile-switching function.

First, eight profiles on the DDS must be programmed to test full profile-switching capacity. We use DDS1 on the board as the test unit, and connect 3 TTL lines to it through the FPGA. In the Python script, we program the DDS1 profiles as following:

DDS1 Profile 0: Frequency=10MHz Amplitude=10 Phase=0

DDS1 Profile 1: Frequency=10MHz Amplitude=20 Phase=0

DDS1 Profile 2: Frequency=10MHz Amplitude=30 Phase=0

DDS1 Profile 3: Frequency=10MHz Amplitude=40 Phase=0

DDS1 Profile 4: Frequency=10MHz Amplitude=50 Phase=0

DDS1 Profile 5: Frequency=10MHz Amplitude=60 Phase=0

DDS1 Profile 6: Frequency=10MHz Amplitude=70 Phase=0

DDS1 Profile 7: Frequency=10MHz Amplitude=80 Phase=0

VGA amplitudes all set to 100.

To control the output, we set the TTL lines such that the DDS switches from profile 0 all the way to profile 7, one profile at a time, returning to profile 0 between each. The result from the oscilloscope can be seen from Fig 4.3.

The oscilloscope graph matches our previous prediction of the DDS output from Fig 3.19. The output meets our expectation except for some glitching at the moment of profile change. The ringing is nearly an instant phenomenon, meaning that it has very high frequency. We have not modeled the frequencies in the ringing area, but it appears that the ringing frequency is of a different order of magnitude than

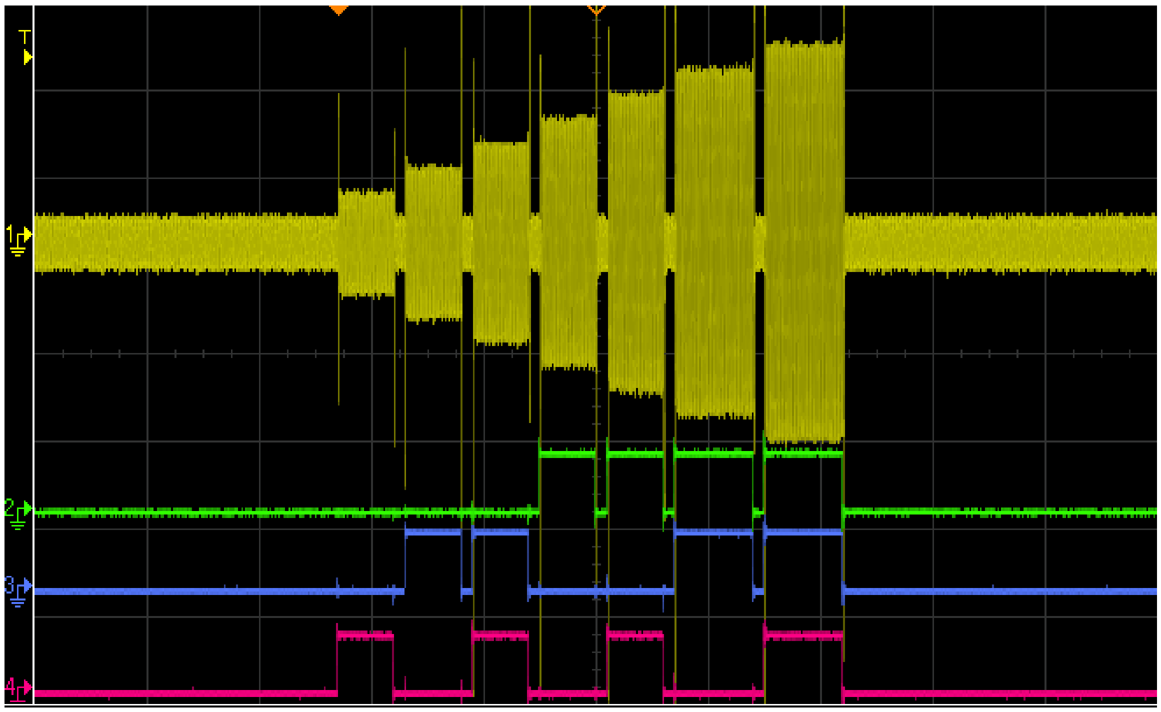


Figure 4.3: DDS output based on our profile settings and the TTL lines. Channel 1 is the DDS output. Channel 2, 3, and 4 are bit 0, bit 1, and bit 2 respectively in the profile selection command. Each grid on the x-axis is 10 microseconds.

the frequencies of interest (0-400MHz). The ringing is also very short in duration– approximately equal to or less than one radiofrequency cycle, so the the possibility that it will cause disturbance to the cooling process is negligible.

We also wish to look at the area of transition carefully to see if any delay is present since it could cause a phase error. We zoom in on the oscilloscope to observe the region where the DDS output updates to a new profile. Fig 4.4 shows the oscilloscope graph. The first thing to notice is a slight ringing in the transition– we can see the cycle in the center has slightly bigger amplitude than the adjacent cycles. We can also see that all the cycles are in phase each other. Although we performed a profile switching at this point, the DDS still makes sure there is no phase shift during the transition. Another oscilloscope graph showing a frequency change can be seen in Fig 4.5.

Having known the DDS's actual output capacity, we need to find if the DDS output is consistently producible. We can repeat the pulse sequence simply by running the Labview program that controls the TTL board repeatedly. This generates identical TTL lines, and thus should trigger exactly the same DDS output. Unfortunately, during certain runs we encounter inconsistencies in the output. One example of output inconsistency can be seen in Fig 4.6.

In Fig 4.6, profile 4 replaces profile 0 as the final pulse in the sequence. Keep in mind that this is only an example of output error. We have repeated the generation many times, and almost every pulse in the sequence could indicate a wrong profile switch. In the example given, the TTL lines send command 000 to the DDS, but the DDS perceives it as 100, corresponding to the 5th profile. In this case, TTL1 seems to be the culprit, since its logic level is regarded by the DDS as high while it should be low.

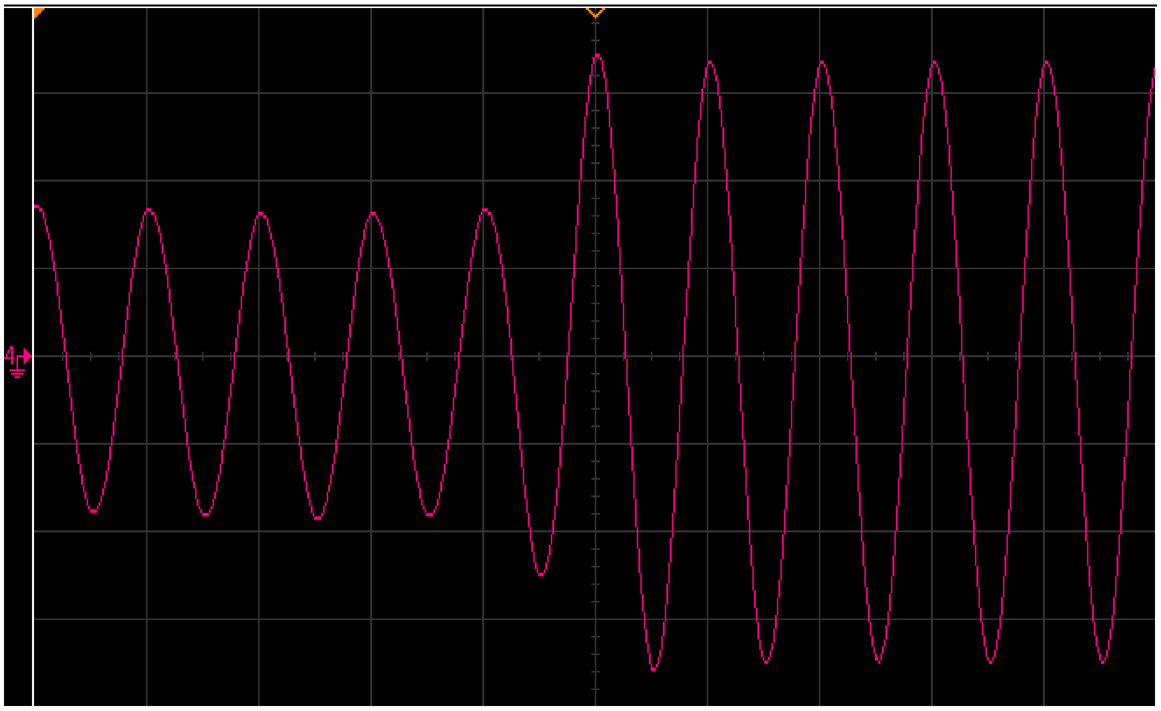


Figure 4.4: The region of the transition from one profile to another. The central cycle has slightly higher amplitude. Despite the existence of this transition cycle, the two waveforms are in phase with each other. Each grid on the x-axis is 10 nanoseconds.

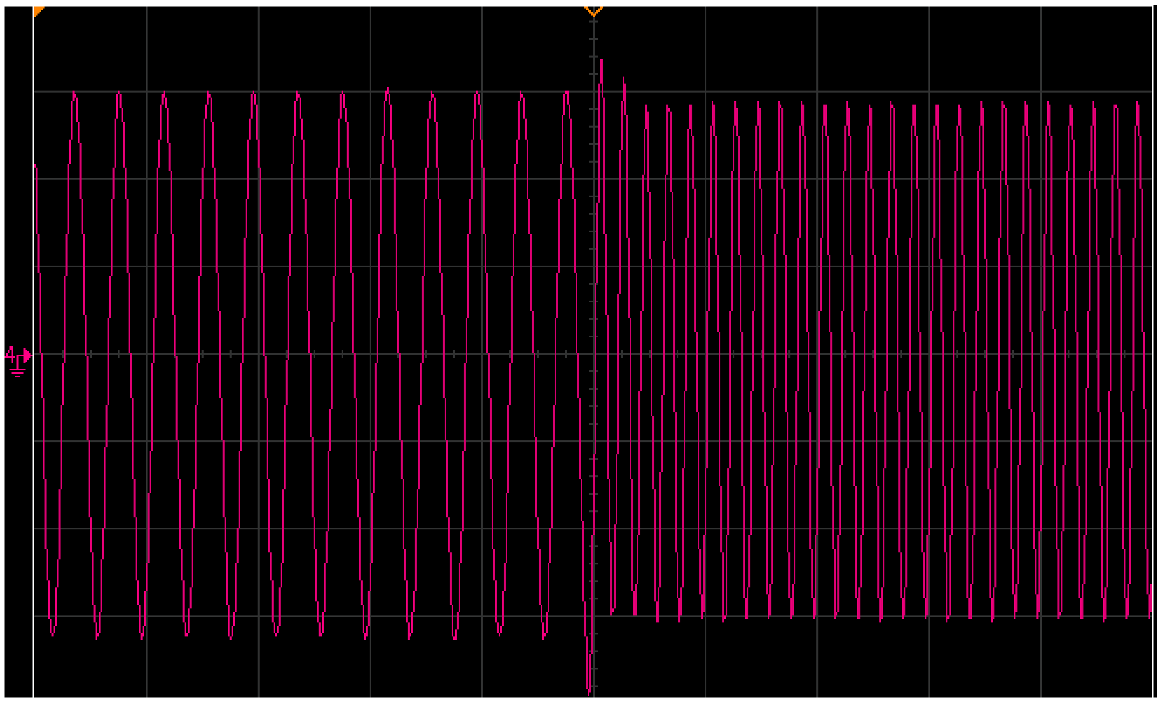


Figure 4.5: The region of the transition from one profile to another. This time the frequency is set to double after the profile change. Each grid on the x-axis is 50 nanoseconds.

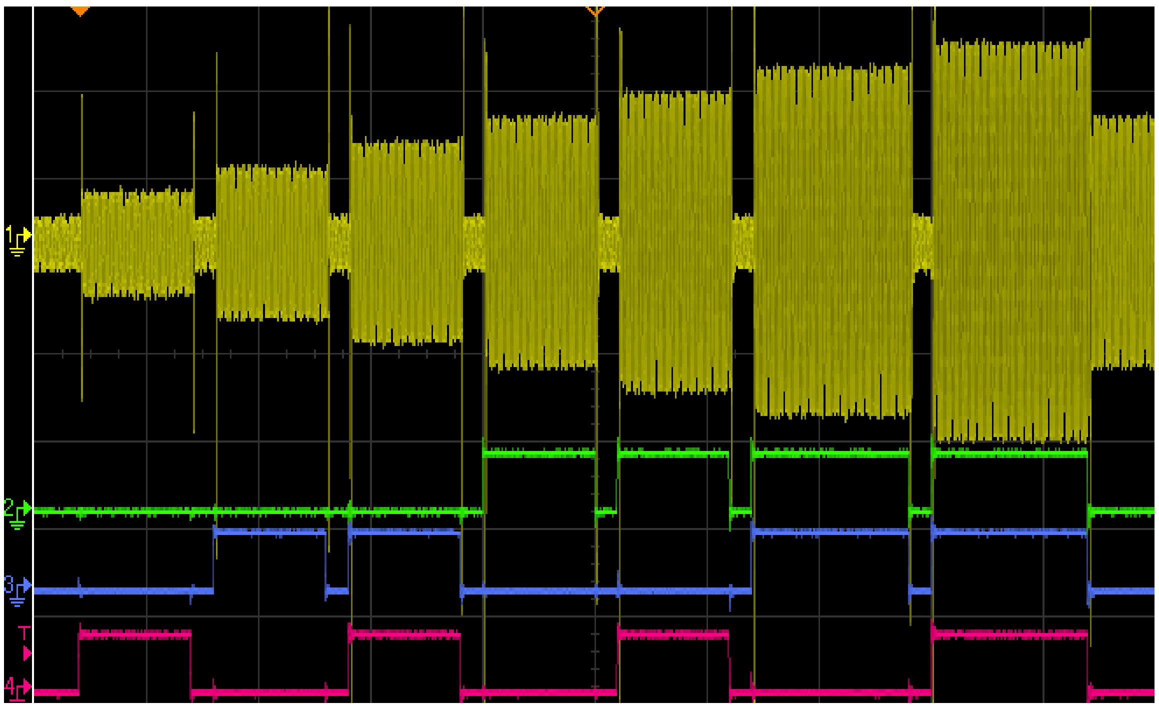


Figure 4.6: An example of the DDS output involving an incorrect pulse in the sequence. After the profile 7, the DDS should supposedly switch to the profile 0 since the TTL lines give 000 as the profile selection, but it in fact switched to profile 4 (100). Each grid on the x-axis is five microseconds.

We need to understand the AD9910's profile switching mechanism before we can tell what is causing the output inconsistency. The DDS uses a synchronization clock (SYNC_CLK) signal to time the transition from one profile to another. SYNC_CLK is obtained from the system clock (1GHz) by diving it by 4 using a frequency divider. A rising edge of SYNC_CLK initiates a transfer of data from the designated profile register to a I/O buffer. When a rising edge on SYNC_CLK arrives at the DDS, the DDS detects the logic levels of the TTL lines and translate that into the profile number, then it retrieves the data in that profile and transfers them to the I/O buffer. The data stored in the I/O buffer are not activated, meaning that they have no effect on the DDS output. The actual activation of the profile is initiated by the I/O update signal, which is sent by the FPGA when it detects a change in the TTL logic levels. After the rising edge of the I/O update signal arrives, the DDS needs to wait for the next rising edge of the SYNC_CLK to initiate the data transfer. The actual transfer happens at the moment when the next SYNC_CLK rising edge arrives. The data transfer sends the data temporarily stored in the I/O buffer to the internal workings (active registers) of the device to generate the actual output. An illustration of the profile switching workflow is given in Fig 4.7. The DDS can automatically update itself every time a set number of SYNC_CLK cycles have passed, so that we repeatedly update the DDS output regardless any change in the TTL bit stream (3-bit command). We do not implement automatic I/O update unless it is proven to be necessary in the future. [18]

Knowing the mechanism of profile switching, we can now investigate the possible cause for output inconsistencies. Imagine having a jitter in one of the TTL lines such that during a certain pulse in that line, the edge on the right falls behind the other

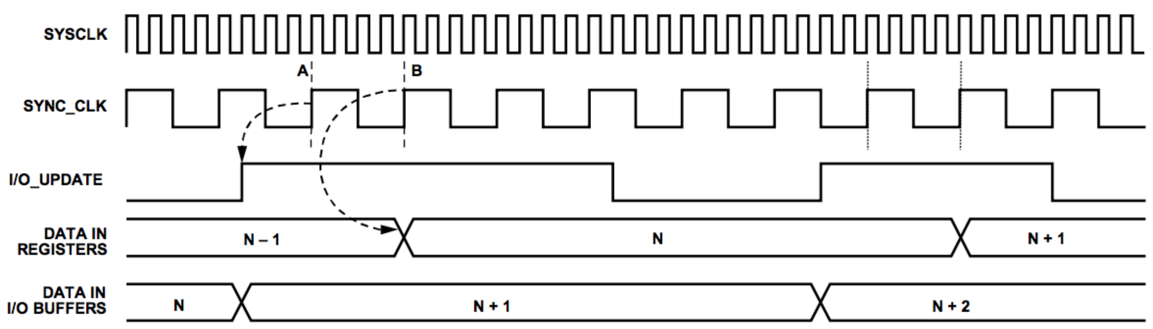


Figure 4.7: Process of profile transfer. Data stored in the I/O buffer indicate the profile selected by the TTL inputs. We see an I/O update pulse is given when the profile data in the buffer is changed due to a change in the TTL logic levels. When the next rising edge of SYNC_CLK arrives at point, the device registers the I/O update, and waits until the next rising edge of SYNC_CLK at point B to initiate the data transfer from the I/O buffer to the internal registers. [18]

two. In this case, the wrong profile data will be stored in the I/O buffer momentarily until the delayed edge arrives, but the FPGA will have already sent out an I/O update signal at that time. Looking at Fig 4.7, we know that this will not cause any trouble if the delayed edge's arrival happens before the actual data transfer (point B), since the device still reads from the I/O buffer, which now contains the corrected profile data. However, if the delayed edge arrives after the I/O buffer data get sent to the internal registers, but before the falling edge of the initial I/O update signal, the device will then transfer the wrong profile data to the internal registers, causing an output error. The delay has to be at least one SYNC_CLK cycle, or $\frac{1}{250MHz} = 4ns$ to allow the actual output to be updated beforehand. See Fig 4.8 for an example of this hypothesis.

To correct this type of error, we could improve our TTL synchronization by implementing better connection solutions. We used unshielded wires to transfer the TTL signals. In the future, we want to find shielded cables with mating connectors on

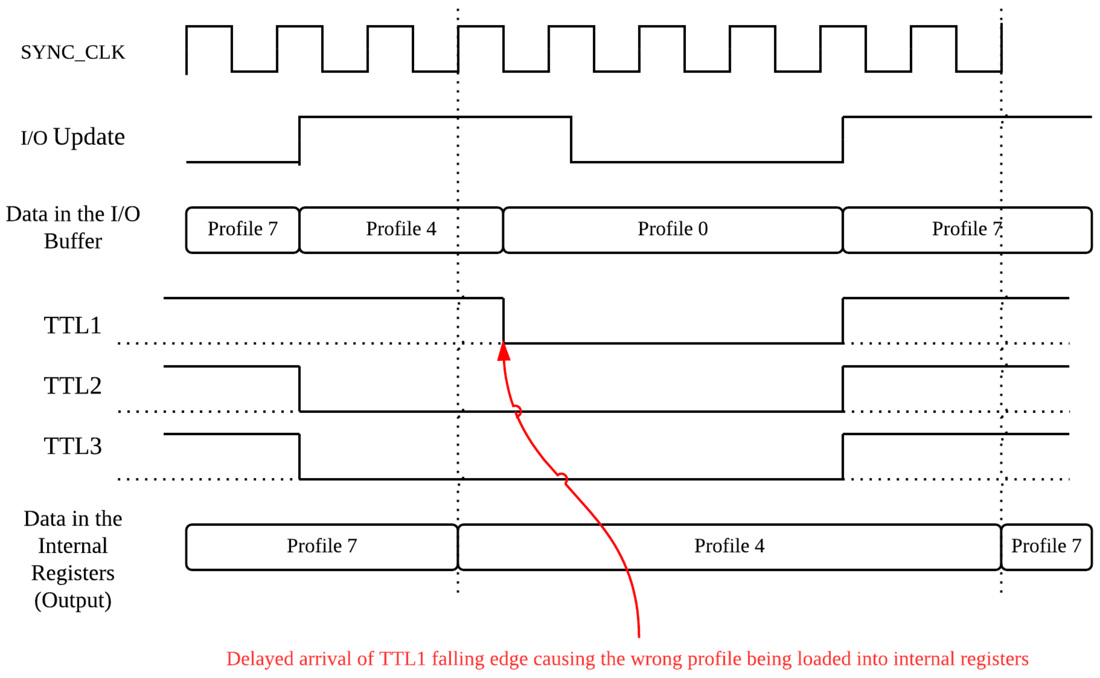

SYNC_CLK
I/O Update
Data in the I/O Buffer
Profile 7
Profile 4
Profile 0
Profile 7
TTL1
TTL2
TTL3
Data in the Internal Registers (Output)
Profile 7
Profile 4
Profile 7
Delayed arrival of TTL1 falling edge causing the wrong profile being loaded into internal registers

Figure 4.8: An example of the output error caused by a TTL delay. The I/O update signal initiates a data transfer to the internal registers before the wrong profile in the I/O buffer is corrected, causing a continued output error. The output will resume back to normal in the next profile switch, provided this type of error does not happen again.

each end to link the TTL board to the DDS board and backplane headers. Shielding and mating connectors can provide protection against noise acquired from external sources, and may reduce the the delay of TTL lines. We may also improve the TTL rising and falling edges, where ringings are present, by performing impedance matching on TTL cables.

Chapter 5

Conclusion

We have constructed two separate radiofrequency generation systems that, when used in conjunction, may effectively create a pulse sequence that meets the requirements for modulating the laser beam in the two cooling schemes. In the actual experiment, we need to configure the system based on the actual pulse sequences found by going through the theories presented in Chapter 2. The complete cooling sequence, although short in time, may have a large number of pulses. In addition, having to run multiple channels simultaneously added extra tediousness to system control. In the long run, we wish to develop a comprehensive method to realize automatic control of the system based on the condition information we provide– such as the ground-state cooling of a $^9\text{Be}^+$ ion. A future researcher in our lab, even with little knowledge on the instruments discussed in this thesis, is expected to be able to use the system to create a modulation pulse sequence efficiently.

One of the goals for the near future is solving the cabling and connecting problems as the external noise reduce the precision of our output signal. We may also want to optimize the computerized control softwares for easier controllability. After that we will be ready to install the acousto-optic modulator and modulate the laser for the ground-state cooling once our apparatus is completed.

Appendix A

Programming the DDS Profile Registers

This appendix shows how to use the Python script to program the DDS profile registers. The script realizes profile register and VGA control by using a finite state machine. As we covered in Chapter 3, we have payload commands and VGA settings programmed by this script. To begin with, let us look at the commands we have:
wr_msb, wr_csb, and wr_lsb are the 24-bit control words that write into a 72-bit register on the FPGA, and later into the designated DDSes.
set_vga is a 32-bit command that sets the VGA (external) value, and I/O updates the DDSes' outputs.

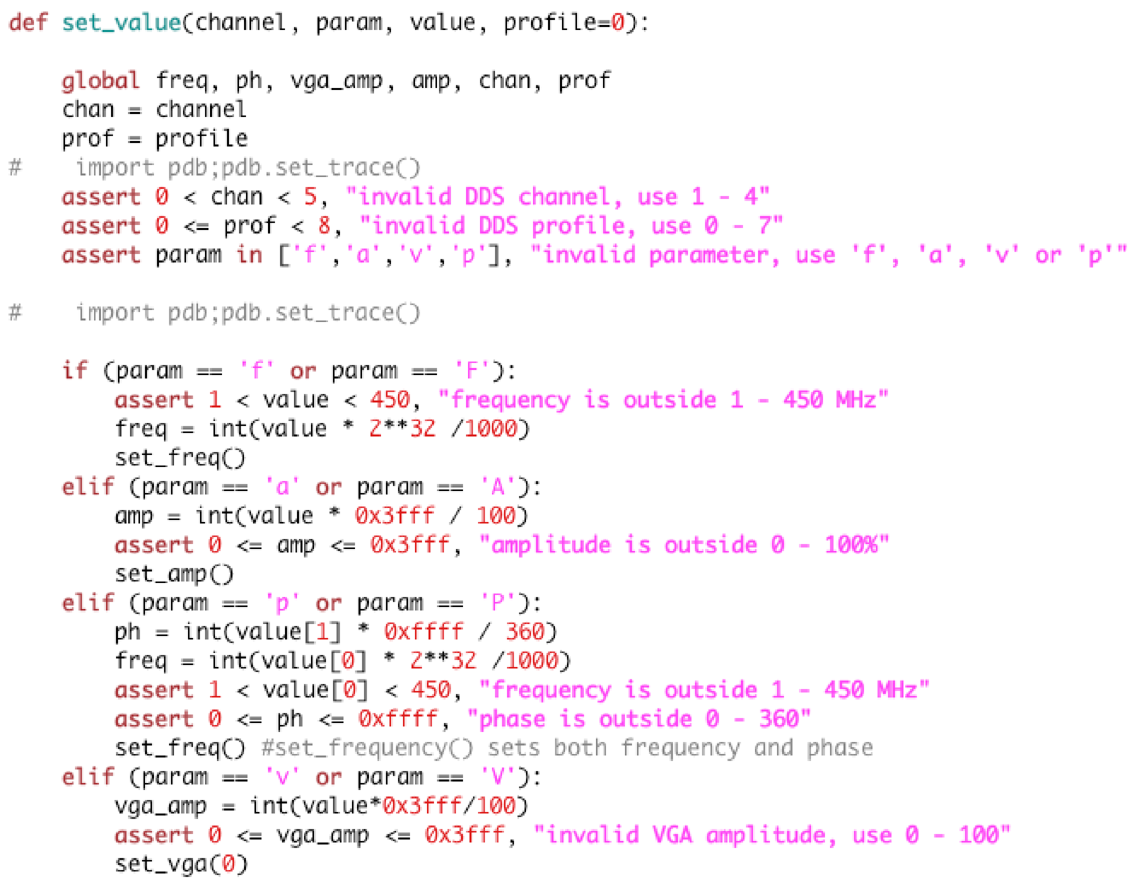
We write instructions in a text file as following:

```
1 a 000 f 075 v 100 0
1 a 007 f 075 v 100 1
2 a 000 f 080 v 100 0
```

```
2 a 100 f 080 v 100 1
```

The first number in line is the DDS number (1-4). a means amplitude in percentage. f means frequency in MHz. v means VGA amplitude in percentage. The last number in each line is the DDS profile that can be anything from 0 to 7. We can also implement phase control by using p commands with its value in 0-360 (degrees).

The Python script reads these definitions and translate them into system commands by using the set_value instruction, as seen in A.1. We may not use frequency higher than 400MHz due to the instrument limitation. We can see here the VGA setting and the DDS amplitude setting give us dual power control, which can come in handy if at a later stage we find attenuation of the output signal necessary.



```python
def set_value(channel, param, value, profile=0):

    global freq, ph, vga_amp, amp, chan, prof
    chan = channel
    prof = profile
#    import pdb;pdb.set_trace()
    assert 0 < chan < 5, "invalid DDS channel, use 1 - 4"
    assert 0 <= prof < 8, "invalid DDS profile, use 0 - 7"
    assert param in ['f','a','v','p'], "invalid parameter, use 'f', 'a', 'v' or 'p'"

#    import pdb;pdb.set_trace()

    if (param == 'f' or param == 'F'):
        assert 1 < value < 450, "frequency is outside 1 - 450 MHz"
        freq = int(value * 2**32 /1000)
        set_freq()
    elif (param == 'a' or param == 'A'):
        amp = int(value * 0x3fff / 100)
        assert 0 <= amp <= 0x3fff, "amplitude is outside 0 - 100%"
        set_amp()
    elif (param == 'p' or param == 'P'):
        ph = int(value[1] * 0xffff / 360)
        freq = int(value[0] * 2**32 /1000)
        assert 1 < value[0] < 450, "frequency is outside 1 - 450 MHz"
        assert 0 <= ph <= 0xffff, "phase is outside 0 - 360"
        set_freq() #set_frequency() sets both frequency and phase
    elif (param == 'v' or param == 'V'):
        vga_amp = int(value*0x3fff/100)
        assert 0 <= vga_amp <= 0x3fff, "invalid VGA amplitude, use 0 - 100"
        set_vga(0)
```

Figure A.1: The part of the Python script that translate our definitions of the waveforms into system control words.

Appendix B

TTL Connections

The Milldown DDS board features two 10-pin headers to receive logic commands. In addition, there are headers on the backplane, one for each DDS board. for We can use them for profile switching, or other functions such as powering down the board, or manual I/O update.

In Fig B.1, we can see the two headers on the DDS board. Grouped into CONNA and CONNB, as seen in Fig B.2 the logic commends from the two headers are sent to the FPGA (top-left). Note that we have eight lines in each group at maximum.

The routing for the commands coming from the backplane connector is a bit more complicated. The connector itself can be seen in Fig B.3. The eight lines on it are labeled P1-4 and N1-4. Tracking those to the DDS connector, these lines are sent to IO 1-4_P/N as seen in Fig B.4.

Fig B.5 shows that they arrive at the DDS board, and are labeled as EXP_IO1-4_P/N. Then, they are grouped into EXP_IOBUS1-8 in Fig B.6. Finally, these lines from the backplane are collected by the FPGA (right), as seen in Fig B.7.

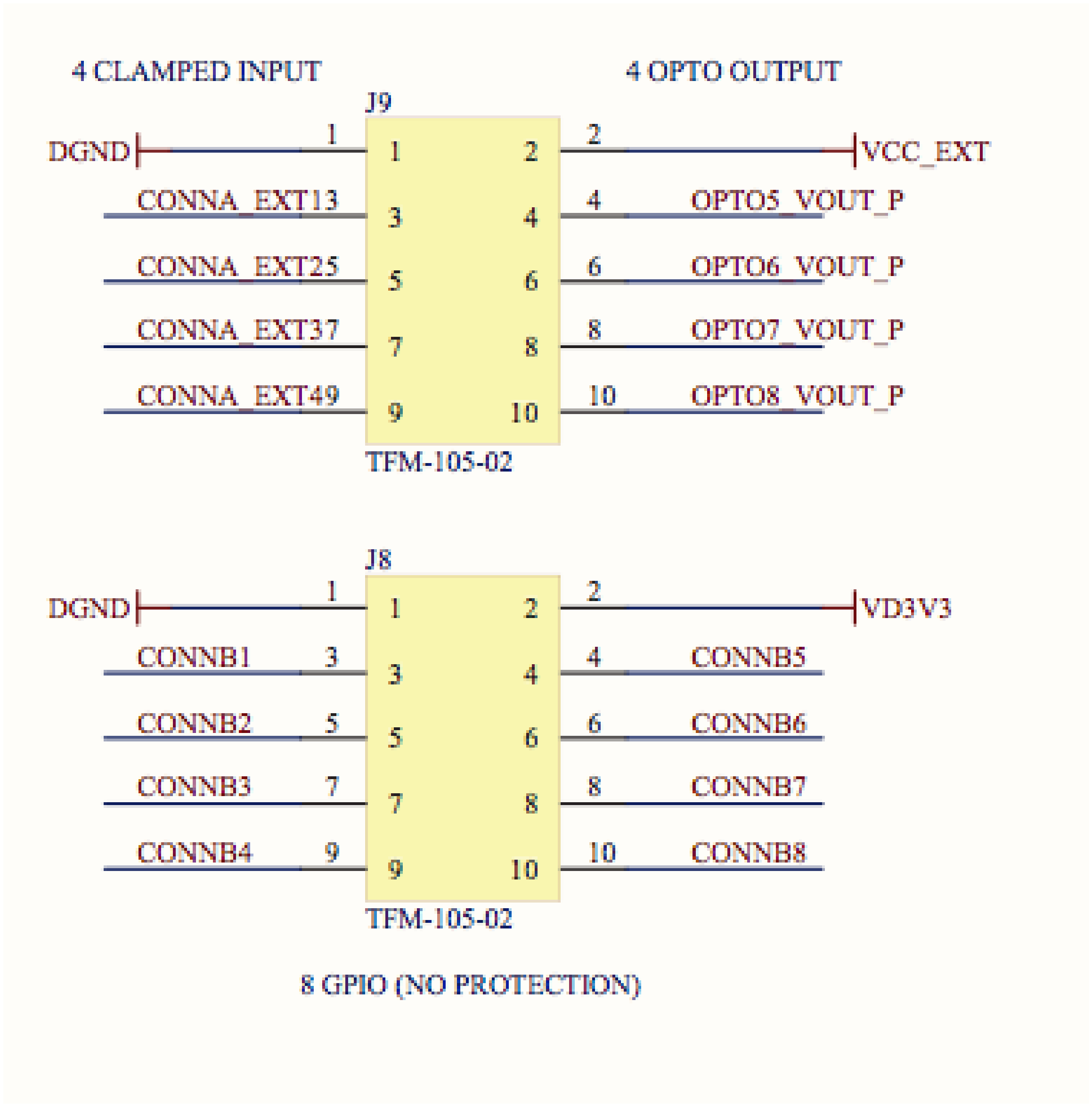


Figure B.1: The two headers on the DDS board. [18]

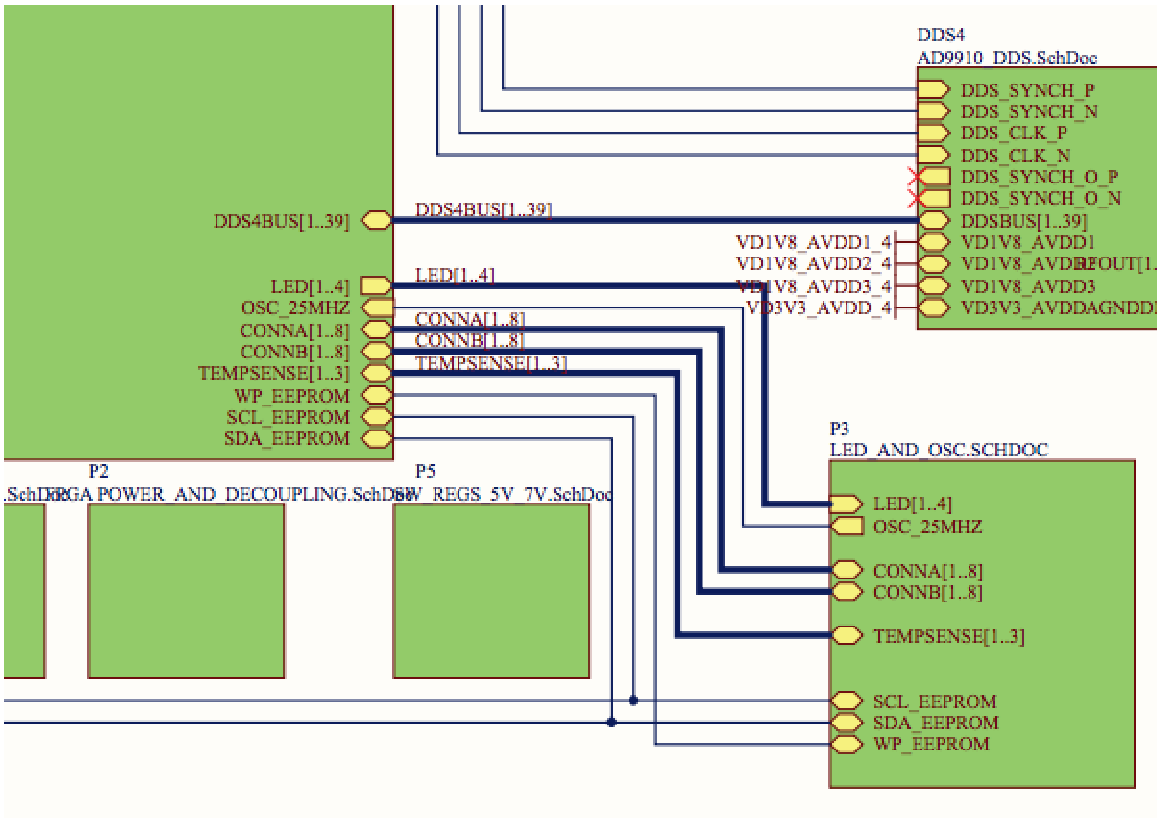


Figure B.2: Arrival at the FPGA. [18]

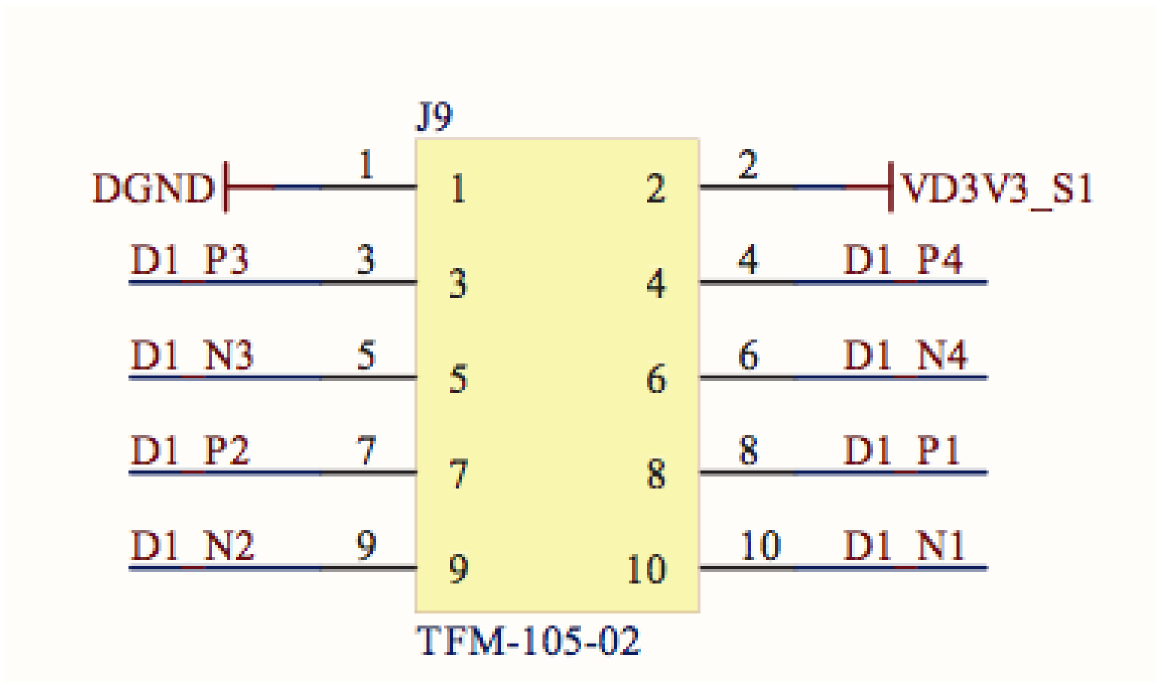


Figure B.3: Header on the backplane. [19]

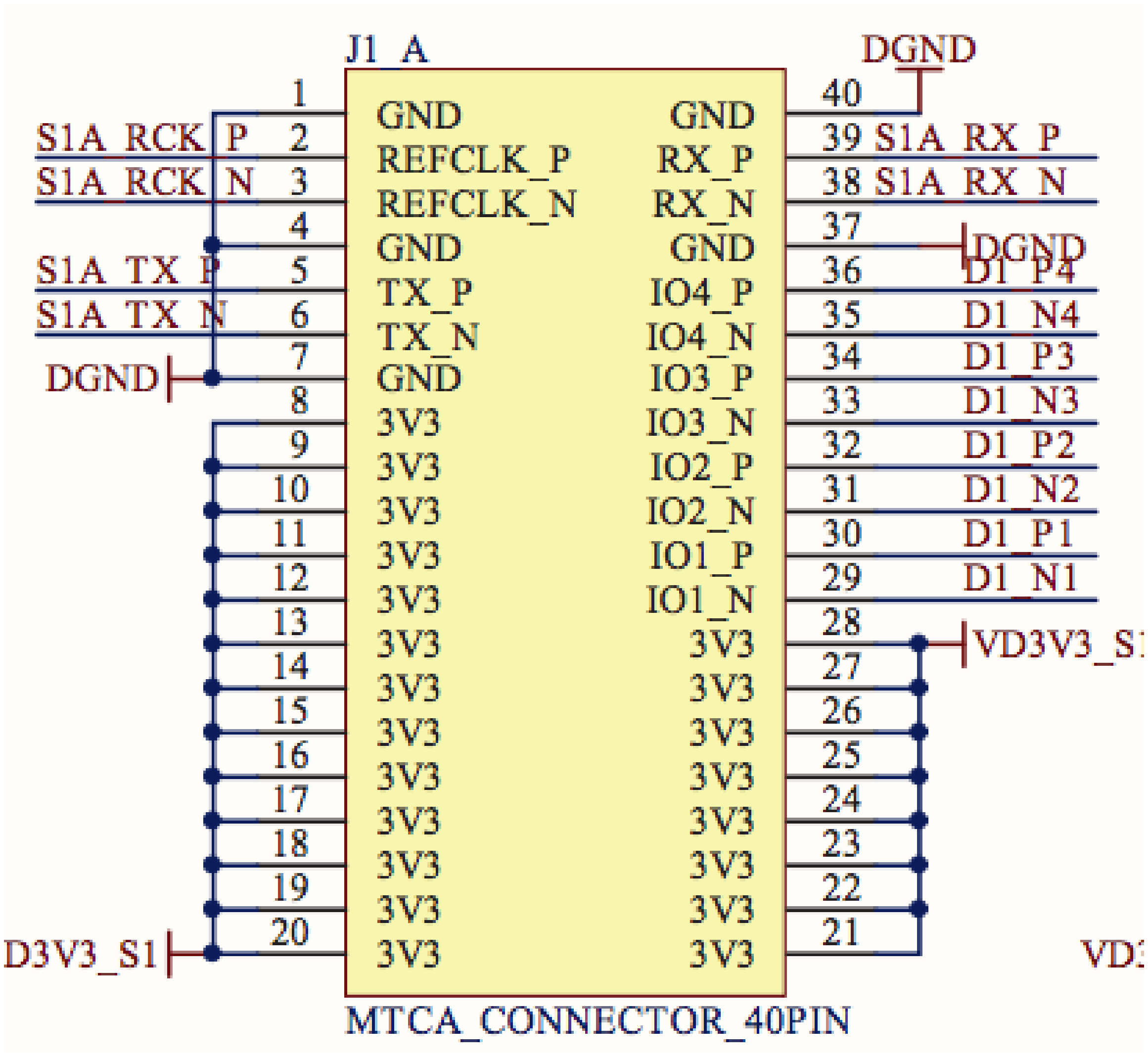


Figure B.4: The backplane lines route to the connector that links the DDS board and the backplane. [19]

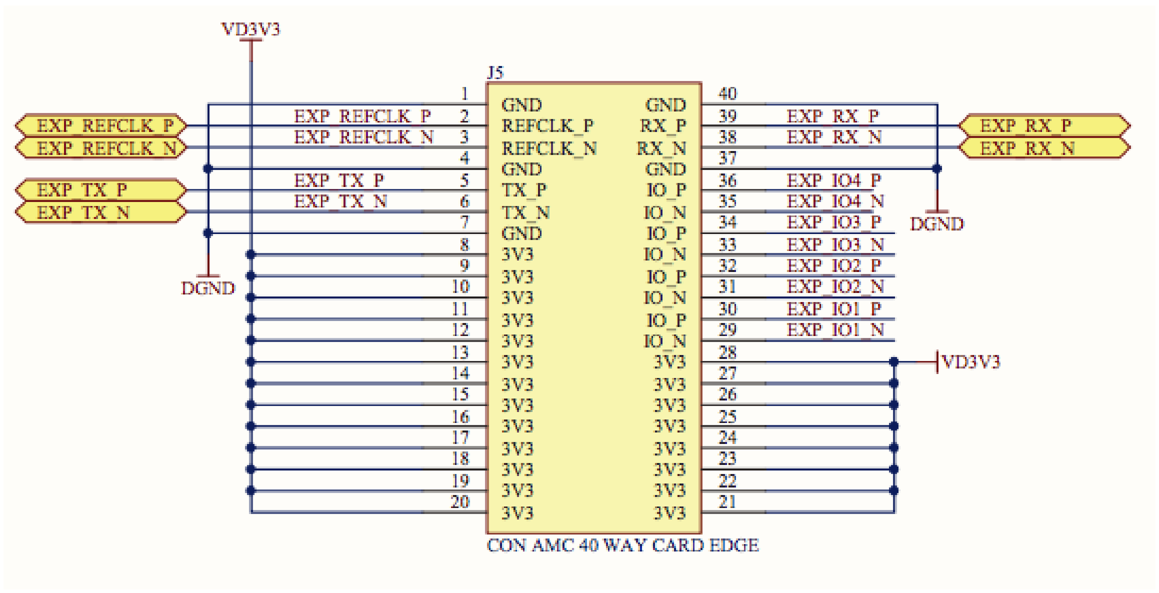


Figure B.5: The lines from the backplane go to the DDS board. [18]

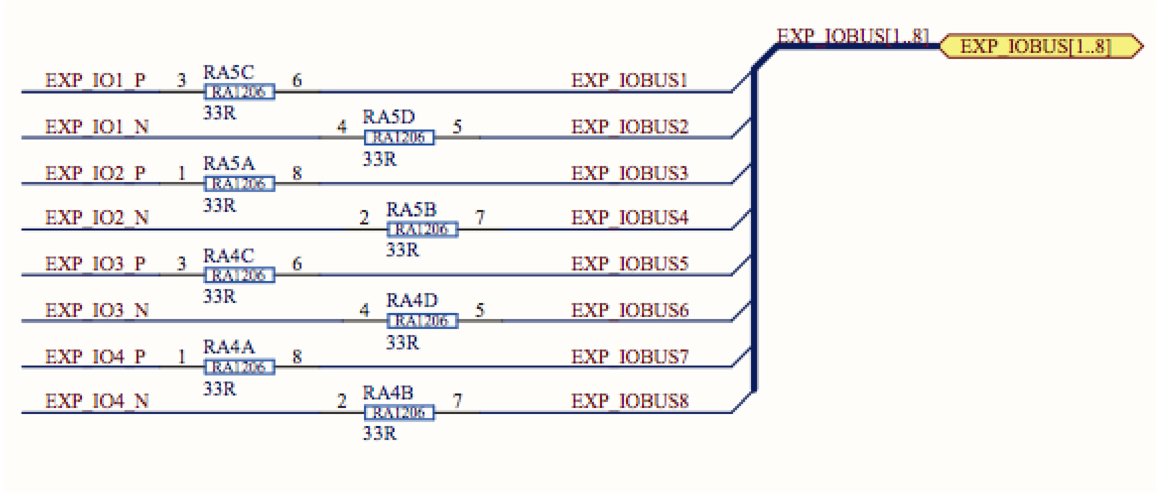


Figure B.6: Grouping of the backplane lines. [18]

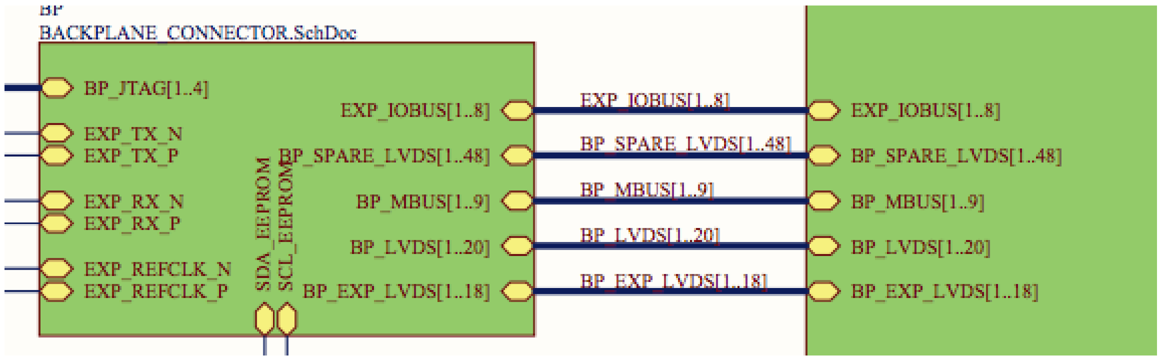


Figure B.7: Arrival at the FPGA [18]

Appendix C

Programming the FPGA

We to alter the FPGA firmware, so that it can direct our TTL lines to the DDS as desired. The firmware is written in Verily, a hardware description language. We ought to change only the part that tells the FPGA the assignment of each TTL line. Looking at the firmware, we can see: assign dds1_profile_o = x, y, z;
assign dds2_profile_o = x, y, z;
assign dds3_profile_o = x, y, z;
assign dds4_profile_o = x, y, z;

x, y, and z are placeholders– those are the spots where we should put the locations of the TTL pins in. The setting for our system is: (Refer to Appendix B to find where those lines originate.) For DDS1, x=clamped_io_i[1], y=bp_io1_p, z=bp_io3_p.
For DDS2, x=clamped_io_i[2], y=bp_io1_n, z=bp_io3_n.
For DDS3, x=clamped_io_i[3], y=bp_io4_p, z=bp_io2_p.
For DDS4, x=clamped_io_i[4], y=bp_io4_n, z=bp_io2_n.

Appendix D

TTL Board Control

Labview provides powerful NI-DAQmx functions for users to control DAQ-compatible devices, such as PXIe-6536, our TTL board.

Follow these steps to acquire digital output:

1. Create a channel in DAQmx. Specify the physical channel in lines. Then wire task and error out to timing.

2. Specify clock. PXIe-6536 is hardware-timed so simply choose Sample Clock from the drop-down menu. For convenience, create a box for clock rate (unit is Hz) and wire it to timing. Again, wire task and error out to DAQmx Write.

3. To load the waveform data, create a file path input on front panel, then wire it to Retrieve Digital WDT.vi. This VI is crucial for Labview to understand HWS waveform data. Digital waveform should be wired to data input on DAQmx Write. Lastly, wire # of samples in wfm to samples per channel on timing.

4. From DAQmx Write, wire task and error out to DAQmx Start Task. You should choose Digital Wfm 1Chan NSamp from the drop-down menu.

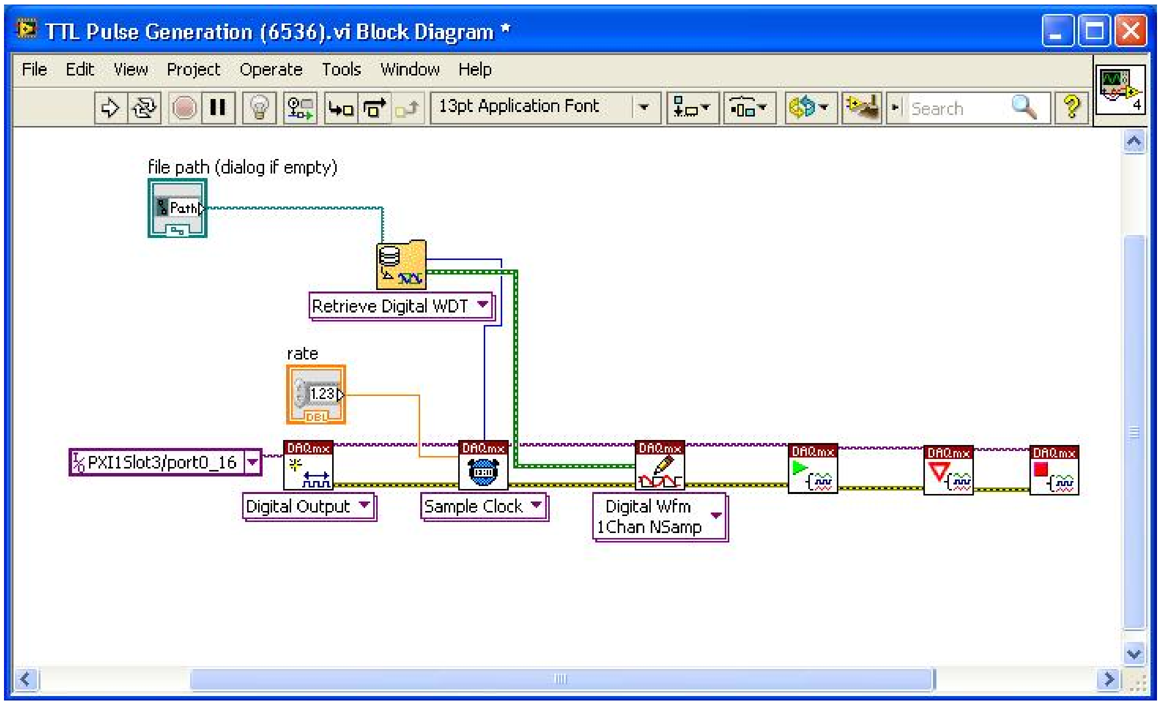


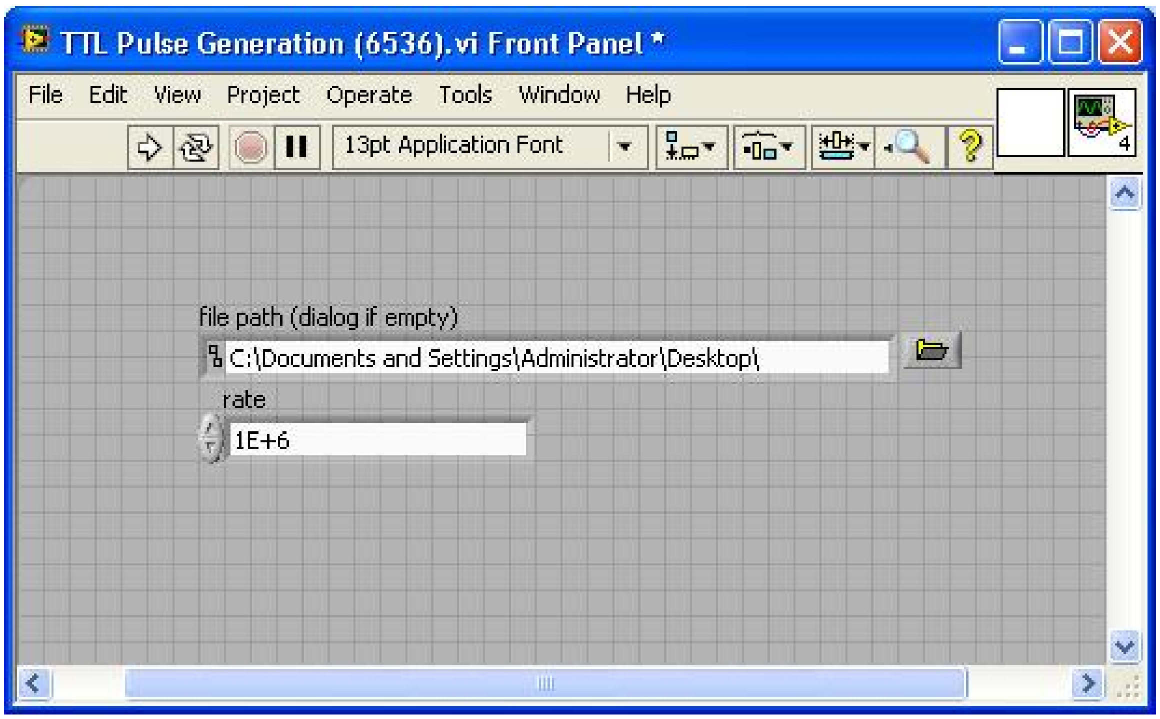
Figure D.1: Block Diagram. Specify physical channel and clock source here.

5. Add error VIs if needed.

See Fig D.1 for the Labview block diagram, and Fig D.2 for the front panel.

Loading HWS waveform data requires the NI-HWS palette. This palette is included in NI-FGEN, NI-HSDIO, or NI-SCOPE. We installed NI-FGEN on our NI box.

With this VI, you can generate pulses based any kind of self-defined waveform data with one click.


TTL Pulse Generation (6536).vi Front Panel *
File Edit View Project Operate Tools Window Help
13pt Application Font

file path (dialog if empty)

C:\Documents and Settings\Administrator\Desktop\

rate

1E+6

Figure D.2: Front Panel. Specify clock rate and HWS file path here.

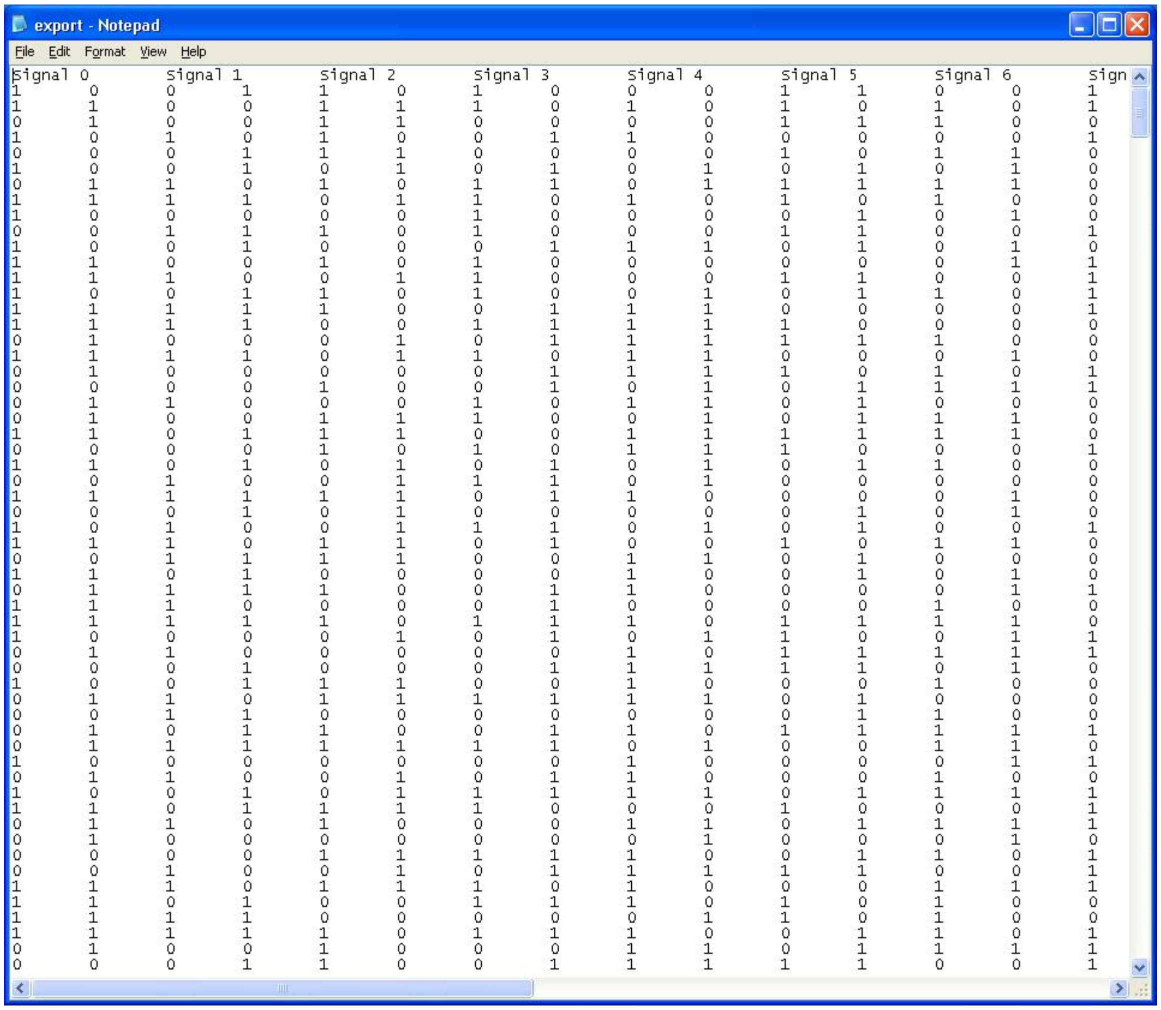
Appendix E

TTL Sequencing

Principally, a pulse sequence can be stored in either a binary file or an ASCII text file. The latter is more convenient since you can see the pattern of the sequence directly. See Fig E.1 for an example text file that defines a pulse sequence. Each number lasts for one clock cycle of the the TTL board.

NI's waveform editor can then convert this text file to HWS file, so that you can load it into the VI in Appendix D.

In long term we would like to develop a software tool to help us create user-defined waveform data. You specify the time periods when the logic level should be high, and the software generates a file based on that.



```
Signal 0            Signal 1            Signal 2            Signal 3            Signal 4            Signal 5            Signal 6            Sign
1         0         0         1         1         0         1         0         0         0         1         1         0         0         1
1         1         0         0         1         1         1         0         1         0         1         0         1         0         1
0         1         0         0         1         1         0         0         0         0         1         1         1         0         0
1         0         1         0         1         0         0         1         1         0         0         0         0         0         1
0         0         0         1         1         1         0         0         0         0         1         0         1         1         0
1         0         0         1         0         1         0         1         0         1         0         1         0         1         0
0         1         1         0         1         0         1         1         0         1         1         1         1         1         0
1         1         1         1         0         1         1         0         1         0         1         0         1         0         0
1         0         0         0         0         0         1         0         0         0         0         1         0         1         0
0         0         1         1         1         0         1         0         0         0         1         1         0         0         1
1         0         0         1         0         0         0         1         1         1         0         1         0         1         0
1         1         0         0         1         0         1         0         0         0         0         0         0         1         1
1         1         1         0         0         1         1         0         0         0         1         1         0         0         1
1         0         0         1         1         0         1         0         0         1         0         1         1         0         1
1         1         1         1         1         0         0         1         1         1         0         0         0         0         1
1         1         1         1         0         0         1         1         1         1         1         0         0         0         0
0         1         0         0         0         1         0         1         1         1         1         1         1         0         0
1         1         1         1         0         1         1         0         1         1         0         0         0         1         0
0         1         0         0         0         0         0         1         1         1         1         0         1         0         1
0         0         0         0         1         0         0         1         0         1         0         1         1         1         1
0         1         1         0         0         0         1         0         1         1         0         1         0         0         0
0         1         0         0         1         1         1         0         0         1         0         1         1         1         0
1         1         0         1         1         1         0         0         1         1         1         1         1         1         0
0         0         0         0         1         0         1         0         1         1         1         0         0         0         1
1         1         0         1         0         1         0         1         0         1         0         1         1         0         0
0         0         1         0         0         1         1         1         0         1         0         0         0         0         0
1         1         1         1         1         1         0         1         1         0         0         0         0         1         0
0         0         0         1         0         1         0         0         0         0         0         1         0         1         0
1         0         1         0         0         1         1         1         0         1         0         1         0         0         1
1         1         1         0         1         1         0         1         0         0         1         0         1         1         0
0         0         1         1         1         1         0         0         1         1         0         1         0         0         0
1         1         0         1         0         0         0         0         1         0         0         1         0         1         0
0         1         1         1         1         0         0         1         1         0         0         0         0         1         1
1         1         1         0         0         0         0         1         0         0         0         0         1         0         0
1         1         1         1         1         0         1         1         1         0         1         1         1         1         0
1         0         0         0         0         1         0         1         0         1         1         0         0         1         1
0         1         1         0         0         0         0         0         1         0         1         1         1         1         1
0         0         0         1         0         0         0         1         1         1         1         1         0         1         0
1         0         0         1         1         1         0         0         1         0         0         0         1         0         0
0         1         1         0         1         1         1         1         1         1         0         1         0         0         0
0         0         1         1         0         0         0         0         0         0         0         1         1         0         0
0         1         0         1         1         0         0         1         1         0         1         1         1         1         1
0         1         1         1         1         1         1         1         0         1         0         0         1         1         0
1         0         0         0         0         0         0         0         1         0         0         0         0         1         1
0         1         1         0         0         1         0         1         1         0         0         0         1         0         0
1         0         0         1         0         1         1         1         1         1         0         1         1         1         1
1         1         0         1         1         1         1         0         0         0         1         0         0         0         1
0         1         1         0         1         0         0         0         1         1         0         1         1         1         1
0         1         0         0         0         0         0         0         0         1         0         0         0         1         0
0         0         0         0         1         1         1         1         1         0         0         1         1         0         1
0         0         1         0         0         1         0         1         1         1         1         1         0         0         1
1         1         1         0         1         1         1         0         1         0         0         0         1         1         1
1         1         0         1         0         0         1         1         1         0         1         0         1         0         0
1         1         1         1         0         0         0         0         0         1         1         0         1         0         0
1         1         1         1         1         0         1         1         1         0         0         1         1         0         1
0         1         0         0         1         0         0         0         1         1         0         1         1         1         1
0         0         0         1         1         0         0         1         1         1         1         1         0         0         1
```

Figure E.1: Logic high level (3.3V) will be 1, and logic low level (GND) will be 0.

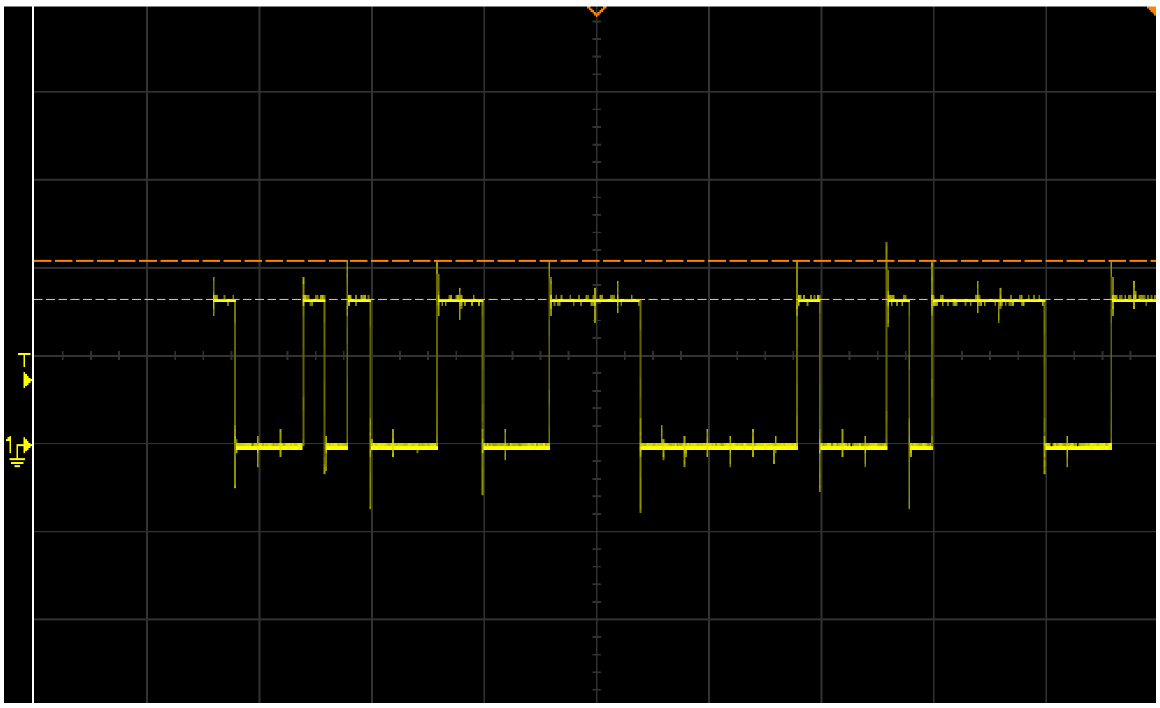


Figure E.2: Example of a TTL sequence based on the definition in Fig E.1. Each grid on the X-axis is five microseconds.

Appendix F

System Enclosure

We built a box to enclose the whole pulse sequence system. The design is simple–the backplane resembles the motherboard inside a computer, so we put it on the bottom. We left enough space on the top, so that the backplane can support up to eight DDS boards. The DDS output ports are SMA connectors. We positioned 32 SMA connectors on the box's front panel for easy connections. All the control signals including USBs and TTLs will go through the back panel of the box. We have two 20-inch fans for two-way cooling, preventing the DDS chips from melting down due to high work temperature. To synchronize all the devices, we use a 8-way splitter to distribute a single reference signal to all the DDS boards. The whole system is powered by a Corsair 760i DC supply unit. See Fig F.1 for the layout of the box, and Fig F.2 for a picture of the system enclosure.

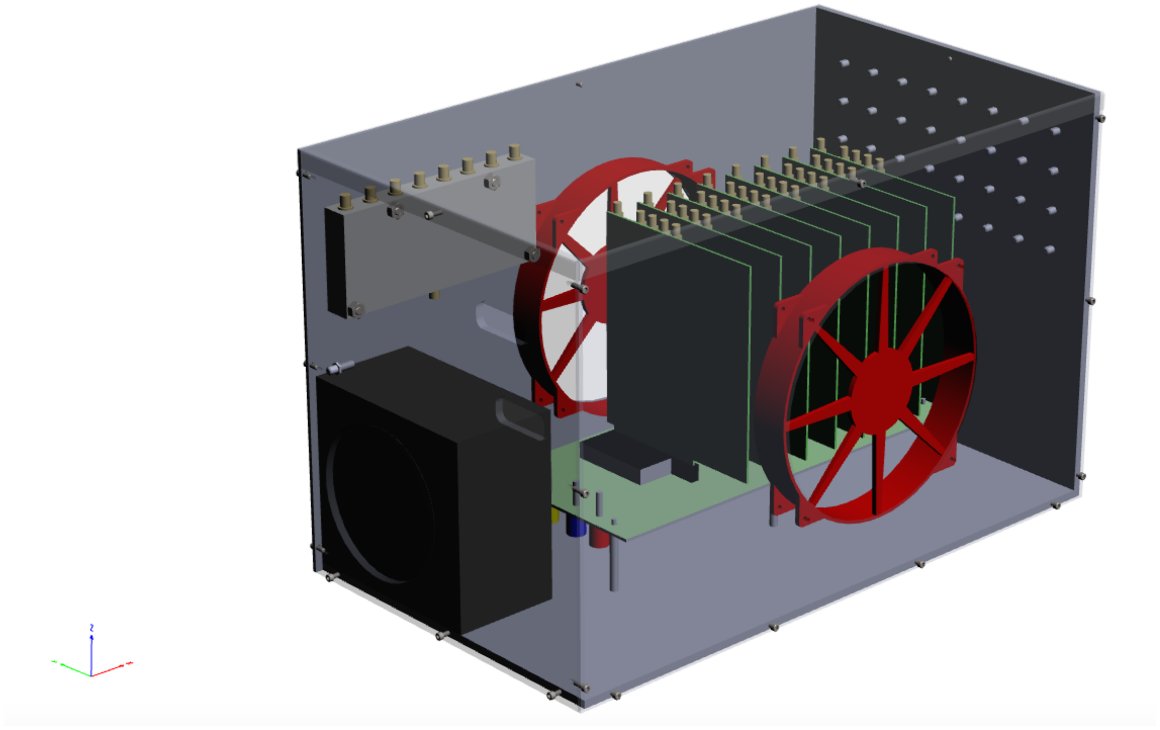


Figure F.1: System enclosure design.

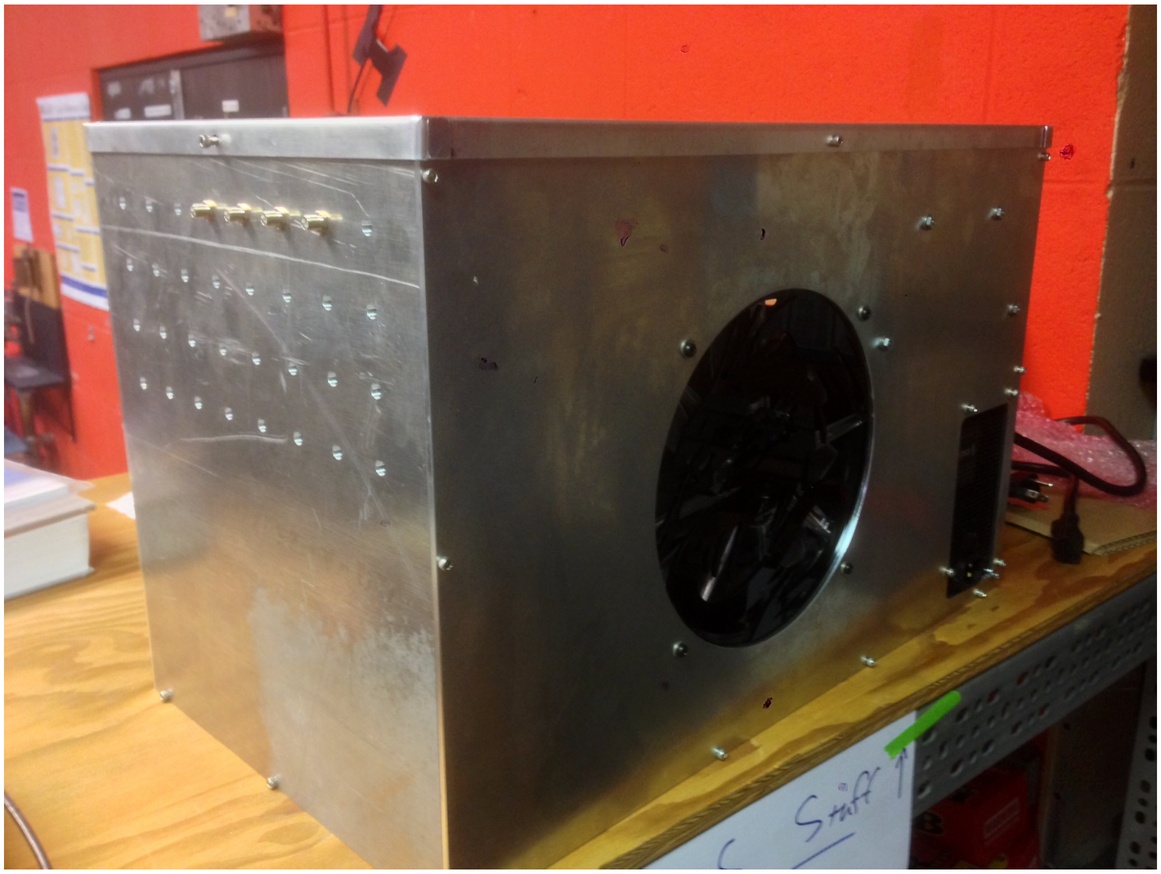


Figure F.2: A picture of the actual box.

Bibliography

[1] M/A-COM Technology Solutions, *MAOC-009269 Voltage Controlled Oscillator*, rev. v2 ed.

[2] Mini-Circuits, *Amplifier ZRON-8G+ ZRON-8G*, rev. a ed. (2008).

[3] W. Marciano, "Time Variation of the Fundamental "Constants" and Kaluza-Klein Theories," Physical Review Letters **52**, 489 (1984).

[4] C. L. W. I. J. B. P. Schmidt, T. Rosenband and D. Wineland, "Spectroscopy Using Quantum Logic," Science **309**, 749 (2005).

[5] S. Qiao, "Constructing a Linear Paul Trap System for Measuring Time-variation of the Electron-Proton Mass Ratio," Amherst College Undergraduate Thesis (2013).

[6] R. B. Blakestad, *Transport of Trapped-Ion Qubits within a Scalable Quantum Processor*, Ph.D. thesis, University of Colorado (2010).

[7] C. Teng, "Frequency Control and Stablization of a Laser System," Amherst College Undergraduate Thesis (2013).

[8] C. J. Foot, *Atomic Physics* (Oxford University Press, 2005).

[9] H. J. Metcalf and P. van der Straten, *Laser Cooling and Trapping* (Springer-Verlag New York, 1999).

[10] J. D. Jost, J. P. Home, J. M. Amini, D. Hanneke, R. Ozeri, C. Langer, J. J. Bollinger, D. Leibfried, and D. J. Wineland, "Entangled Mechanical Oscillators," Nature **459**, 683 (2009), `0901.4779v1`.

[11] D. Leibfried, R. Blatt, C. Monroe, and D. Wineland, "Quantum dynamics of single trapped ions," **75**, 281 (2003).

[12] B. Razavi, *RF Microelectronics* (Paul Boger, 2011), 2nd ed.

[13] J. B. Hagen, *Radio-Frequency Electronics* (Cambridge University Press, New York, 2009), 2nd ed.

[14] Analog Devices, *ADF4108 Data Sheet*, rev. e ed. (2006-2013).

[15] Mini-Circuits, *Wideband Microwave Amplifier AX60-183+*.

[16] Analog Devices, *AD9910 Data Sheet*, rev. d ed. (2007-2012).

[17] Xilinx, *Spartan-6 Family Overview*, 2nd ed. (2011).

[18] Enterpoint Ltd., *Milldown DDS Board Schematics* (2012).

[19] Enterpoint Ltd., *Milldown Backplane Schematics* (2013).