FMCW RADAR

Technical Manual

Frederic Vatnsdal

Telecommunications Project W-16
Presented to: Dr. Raad Al Ashoor
ELECTRONICS ENGINEERING TECHNOLOGY – DAWSON COLLEGE

Table of Contents

Section 1.0 - Abstract

The intended purpose of this Frequency Modulated Continuous Wave (FMCW) Radar system is to capture the position of objects within the radar's operating range and beam width, and display this information on a computer. Data captured by the radar is presented in an analog format. The data is sampled, and then transmitted wirelessly to a receiver module connected to a computer. The computer runs the digital signal processing software, written in Python, which computes frequency and distance based on this data, and displays the positions of the three most significant (based on the strength of the harmonics generated by these objects).

Applications of this system include surveillance systems and drone mounted radar. Thanks to the compact size, low cost, and modularity of the FMCW radar system itself, the radar module can be used in a variety of systems, requiring only a short range radar.

Section 2.0 – System Overview

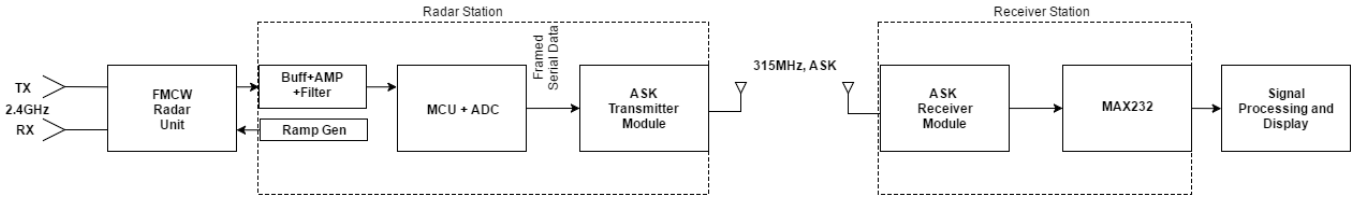


Figure 2.1: System Block Diagram

The system is simplex, with data only flowing from the radar station to the receiver station. The addition of a second link in order to achieve full duplex was provisioned for in the prototype, but ultimately was not implemented in this iteration of the system. The telecommunications link operates at 315 MHz, however, as the link is modular, the transmitter and receiver pair can be replaced by a pair operating at 433 MHz if desired.

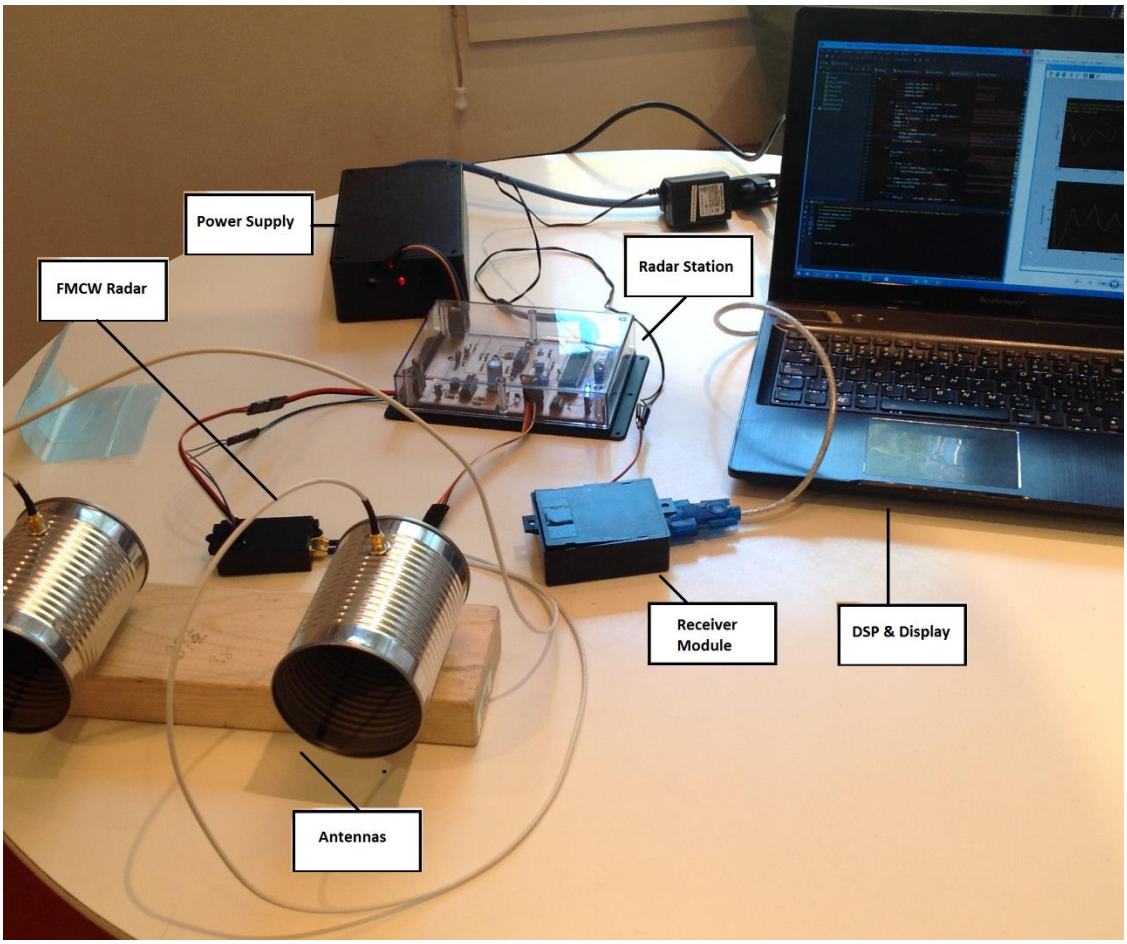


Figure 2.2: Complete system setup.

Table 2.1: System Operating Parameters

Parameter	Minimum	Typical	Maximum	Units
Radar Operating Frequency	2.4		2.5	GHz
Radar Bandwidth	90	100	120	MHz
Radar Power Output		20		dBm
Radar Tune Voltage	0		10	V
Radar Station Power Rails	10, 4.5, 3.0	12, 5, 3.3, 3	12,5.5,3.5,3.5	V
RF Link Operating Frequency	314	315	316	MHz
RF Link TX Power		14		dBm
RF Link Range		10		m
Receive Station Supply Voltage	5	6	35	V

Table 2.1 shows a brief overview of some of the general system operating parameters that can be expected for this system.

Section 3.0 – Installation and Setup

Radar Station Setup Procedure:

1. Connect the radar station power supply to a power outlet.
2. Connect the radar station power supply to the radar station power connector, ensuring that the correct connection is made as shown in figure 3.1.

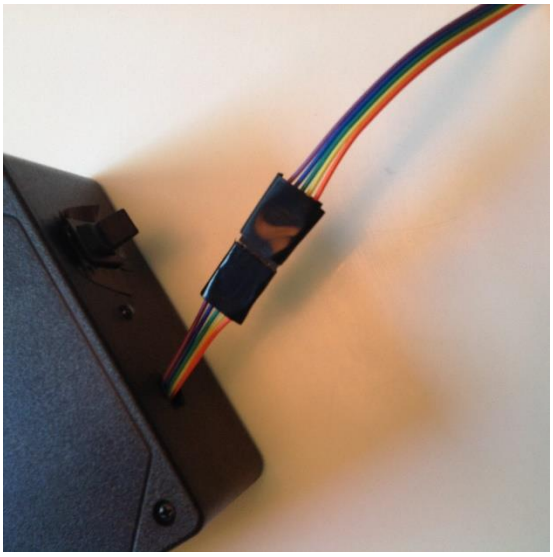


Figure 3.1

3. Connect the radar module to the radar station as shown in figure 3.2. This involves connecting the power connector, and the I/O connector of the radar.

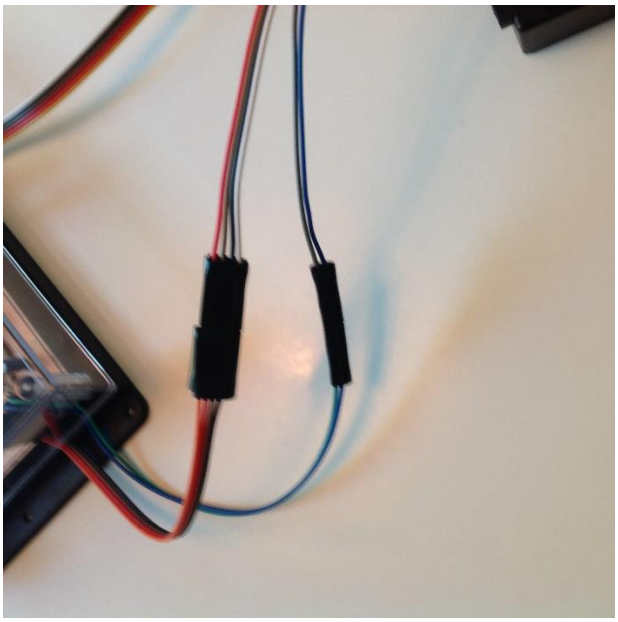


Figure 3.2

4. Deploy the antennas, and connect a coaxial cable to each antenna as shown in figure 3.3.

Figure 3.3

5. Connect one of the antenna coaxial cables to the TX output of the radar, and the other to the RX output of the radar as shown in figure 3.4.

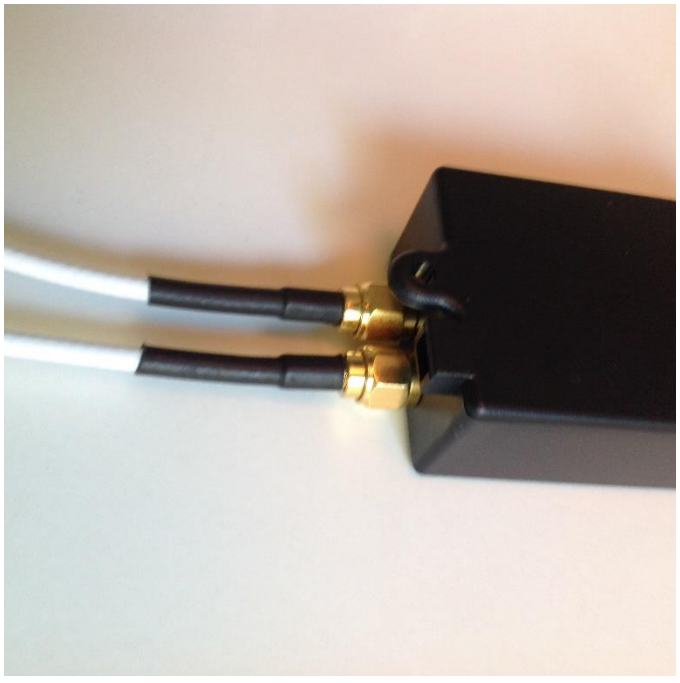


Figure 3.4

6. Set potentiometer R21 (ref figure 5.2) to 0Ω.
7. Power on the system. Ensure power supply LED and the radar station LED illuminate.
8. Using an oscilloscope, monitor test point 4 and adjust the gain of the amplifier using R21 until satisfactory (recommended: 1Vpp output amplitude).

Receiver Station Setup Procedure:

1. Connect the receiver station power supply to an outlet.
2. Connect a serial or USB to serial cable with a male DB9 connector to the receiver's female DB9 port as shown in figure 3.5.

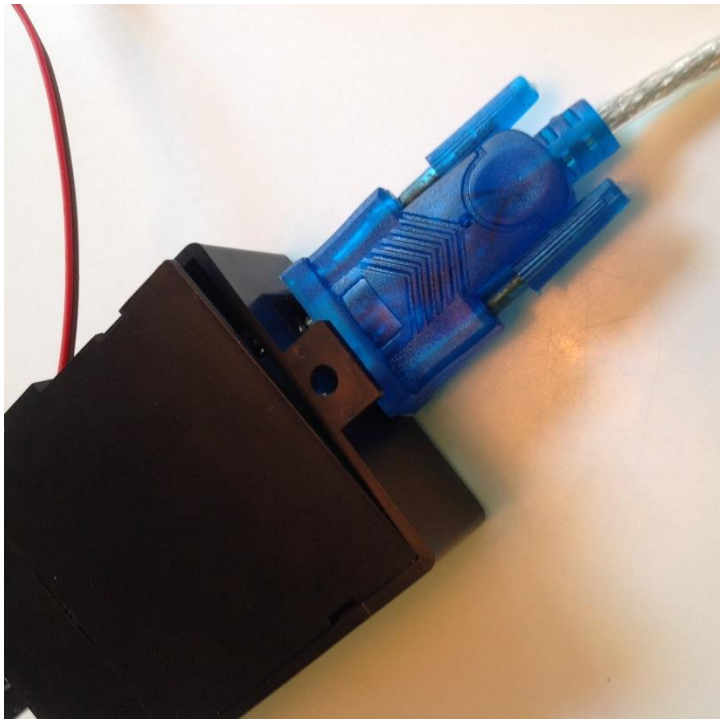


Figure 3.6

3. Connect the other end of the serial cable to the PC.
4. On the PC, run the DSP software.
5. Select configurations from menu (optional)
6. Verify that the time domain and frequency domain plots are visible and updating.
7. Resize the plots as needed.
8. Observe distance results printed to the console for the ranges of objects detected by the radar system.

Section 4.0 - FMCW Radar

The FMCW radar module is the core of this project. It uses a linear ramp input to generate a shifting frequency output at a rate proportional to the frequency of the ramp input, known as a “chirp”. This shift in frequency essentially provides a reference point to compare the reflected wave to. The chirp is mixed with the received signal and the difference of the two frequencies (among other frequencies) is produced. Using the difference in frequency (known as the beat frequency (f_b)), the velocity of the electromagnetic wave, the sweep frequency, and the operational bandwidth of the radar, the distance can be determined. The device was designed to interface with the radar station module. It has one RF and one IF input and one RF and one IF output. The design was intended to be modular, such that it could be used in other systems. Thus only the necessary RF components occupy the radar module PCB.

4.1 - Operation and Design

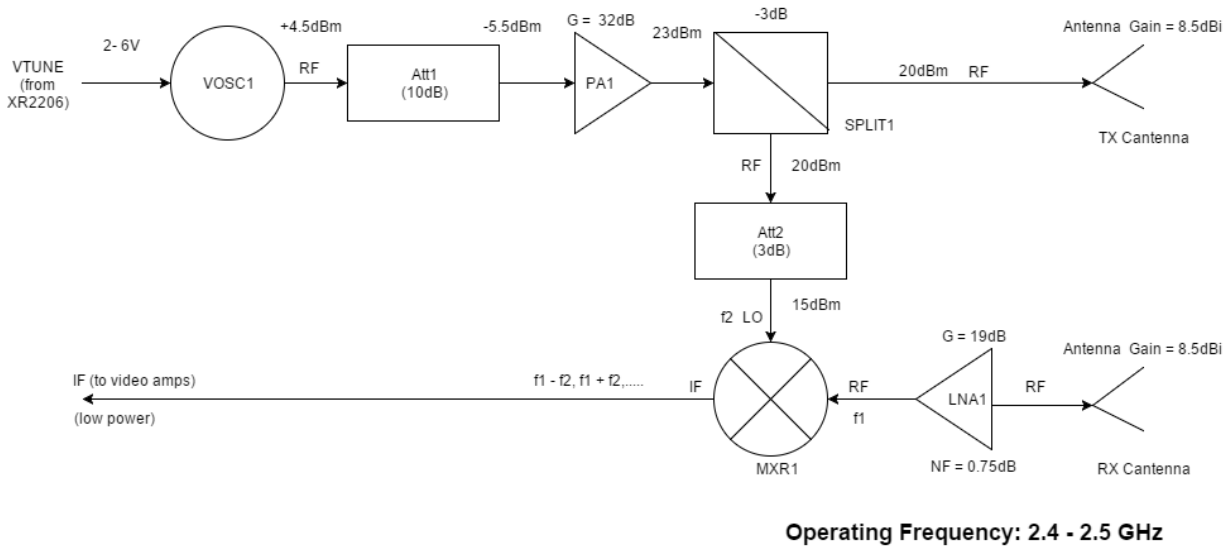


Figure 4.1: Block Diagram of FMCW radar module.

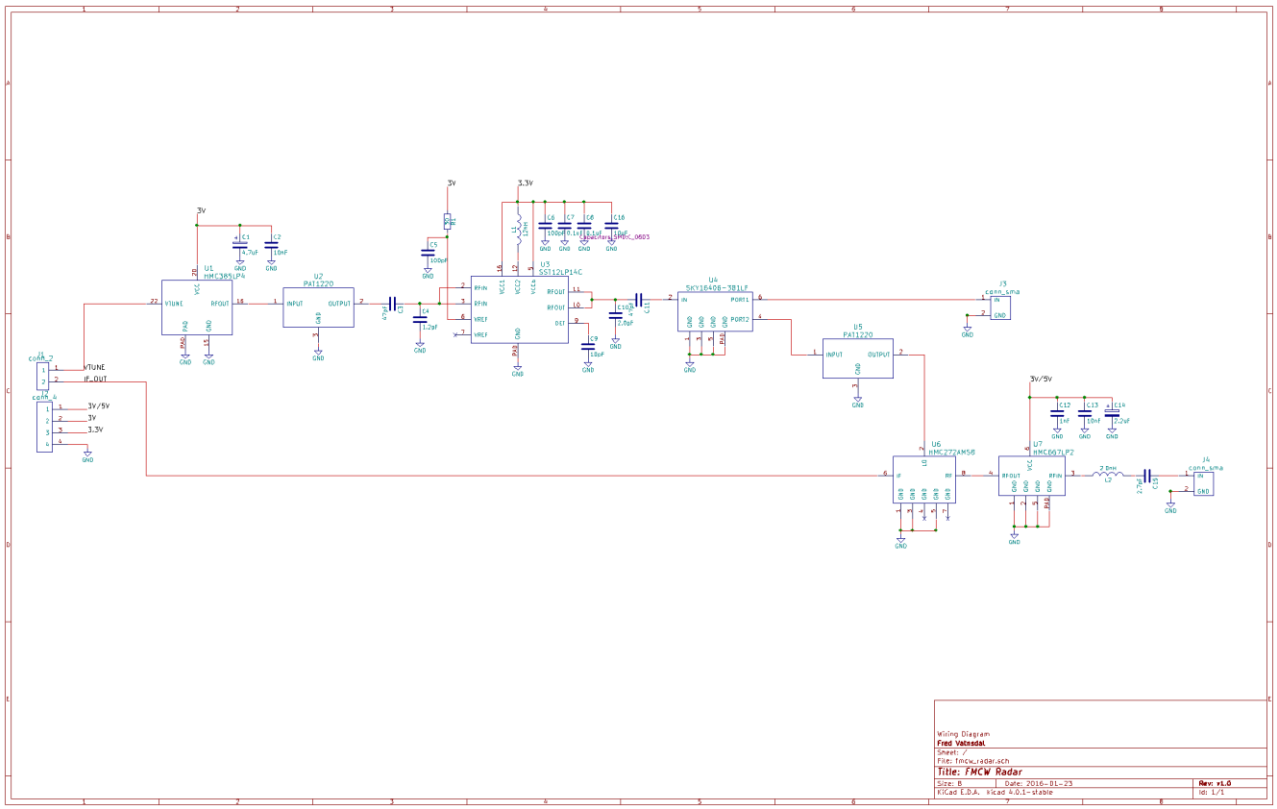


Figure 4.2 FMCW radar module (v1.0) schematic diagram.

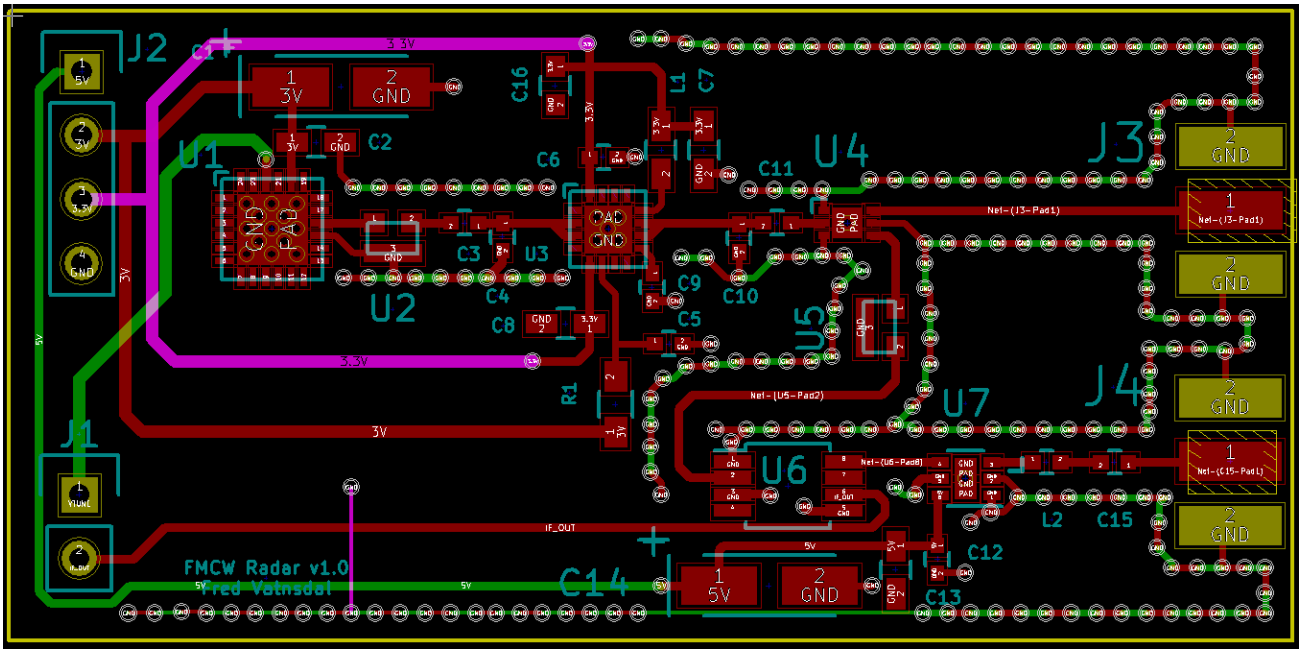


Figure 4.3: FMCW radar module (v1.0) PCB design (without filled areas).

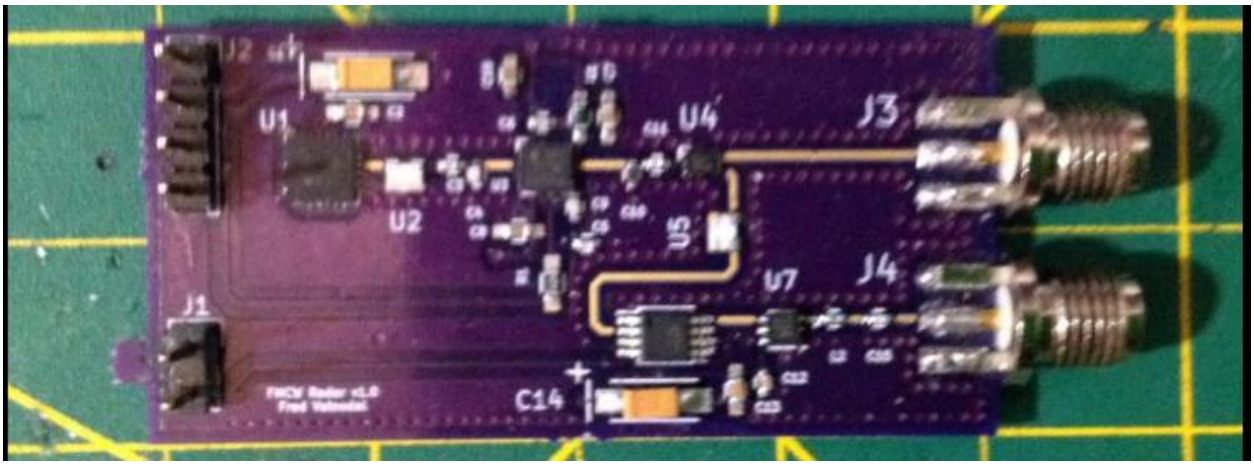


Figure 4.4: Top of the radar module PCB.

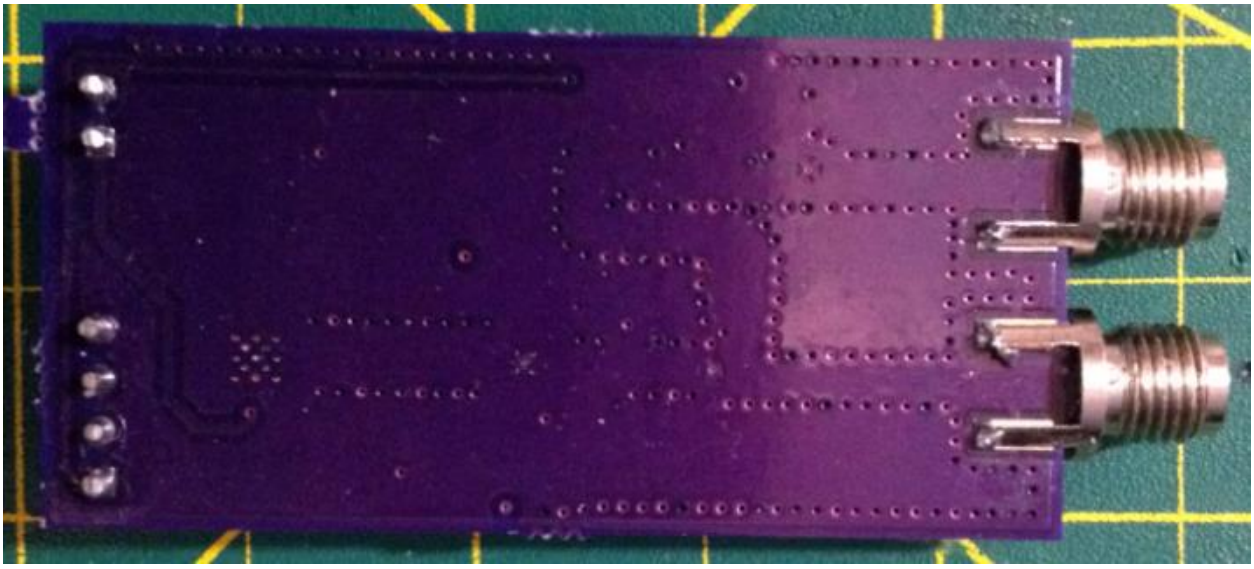


Figure 4.5: Bottom of the radar module PCB.

This particular FMCW radar design is composed of seven RF devices that are specified to operate in the 2.4 to 2.5 GHz range (or a range that encompasses this range). The driven device in this design is the VCO (U1). The VCO chosen for this design was the HMC385LP4. This device was selected, as it operated in the desired frequency band, had a chip package that would be easier to solder compared to other similar devices on the market and had good documentation. The VCO generates the microwave signal at +4.5 dBm. The ramp input to the VCO connects to the first pin of the header J1. Output power of the VCO was not a large design consideration, as the signal will be amplified by a power amplifier before being emitted by the transmitting antenna. The VCO output is first passed through a 10dB passive attenuator (U2). The purpose of this attenuator is to dampen the signal strength, so as to not saturate the power amplifier. The desired attenuator series only had values up to 10dB. This results in -5.5dBm being fed into the power amplifier. In order to prevent saturation, input power would need to be -9dBm. Ultimately, this was a design consideration, as extending the longevity of the system should be a priority if the system were to be manufactured. For the purpose of this experiment/prototype, the pushing the power amplifier into saturation is acceptable, although still not recommended. In order to remedy this, either a VCO with

lower output power or a different series of attenuator could be chosen. The power amplifier (U3) chosen is the SST12LP14C, which is designed to operate in the 2.4 to 2.485 GHz band. That, and its low cost relative to other power amplifiers, made it an excellent choice for the project. Theoretically, the output power of the amplifier should be +20dBm (give or take). The output signal of the power amplifier is split by U4, a Wilkinson power divider, with power being routed to the transmission antenna connected to female SMA connector J3 (reference figure 4.3) and to the local oscillator input of the mixer U6, after passing through the attenuator U5. The splitter chosen is the SKY16406-381F, which is designed to operate in the 2.2 – 2.8GHz range. The reflected wave is captured by the antenna connected to the female SMA connector J4. It is amplified by the low noise amplifier U7. The low noise amplifier chosen was the HMC667LP2, with a gain of 19dB, and a noise figure of 0.75dB. A low noise amplifier, as the received signal could potentially be quite low power, and thus an amplifier that does not contribute a lot of noise is required. The output of the low noise amplifier is fed into the RF input of the mixer, U6. The chosen mixer is the HMC272AMS8. This mixer has an operating frequency range of 2 to 3 GHz. The IF output of the radar is connected to the second pin of the header J1. This mixer output consists of several different frequencies. The frequency of interest is the difference between the local oscillator and RF frequencies, as that is the beat frequency. In order to obtain this frequency (which is a very low frequency, especially in comparison to the other frequencies generated at the output), the low frequency signal must be filtered. This filter is part of the radar station module, and is discussed in Section 5.3.

The FMCW radar v1.0 PCB design is show in figure 4.3. Unlike typical PCB designs, RF designs demand special precautions. One of the most important aspects of the PCB design to consider is the trace impedance of the traces connecting RF components.

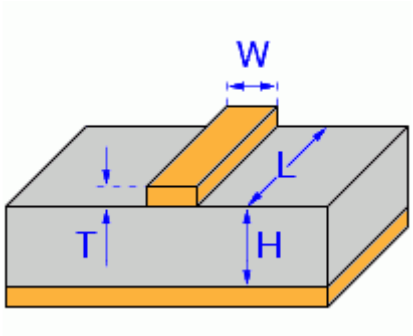


Figure 4.4: Microstrip transmission line. (Source: kicad software calculator)

Figure 4.4 shows a cross section of a microstrip transmission line. In order to calculate the characterisitic impedance of such a transmission line, equation 2.3 can be used.

Equation 4.1:

$$Z_o = \frac{120\pi}{\sqrt{\epsilon_{eff}}} * \frac{1}{\frac{w}{h} + 1.393 + 0.677\ln(\frac{w}{h} + 1.444)}$$

Where:

$Z_o = characteristic\ impedance\ of\ the\ TL$

$\epsilon_{eff} = dielectric\ constant\ of\ substrate\ material$

$w = width\ of\ the\ trace$

$h = height\ of\ the\ substrate\ material\ (spacing\ between\ microstrip\ and\ GND\ plane)$

Equation 4.1 does not take into consideration the height of the trace itself, T, an important parameter. The board that is used for the radar uses an FR408 substrate. At 2GHz it has a dielectric constant, ϵ_{eff}, of 3.67. The spacing between layers that OSH Park, the PCB manufacturer, uses is 0.17mm on a four layer board. The target characteristic impedance of the trace is 50Ω, as that is the specified input/output impedance of the RF components chosen for this design (ideally, there will be perfect matching between source, transmission line, and load).

Assuming a width of ~0.30mm and using equation 4.1:

$$Z_o = \frac{120\pi}{\sqrt{\epsilon_{eff}}} * \frac{1}{\frac{w}{h} + 1.393 + 0.677\ln\left(\frac{w}{h} + 1.444\right)}$$

$$= \frac{120\pi}{\sqrt{3.67}} * \frac{1}{\frac{0.30mm}{0.17mm} + 1.393 + 0.677\ln(\frac{0.30mm}{0.17mm} + 1.444)}$$

$$Z_o = 49.9\Omega$$

Using a width of 0.30mm, the characteristic impedance is determined to be 49.9Ω. This value is close enough to the sought after 50Ω impedance, especially for this prototype RF design. However, using the micro strip transmission line calculator with height of the copper trace (T) taken into account (0.0356mm), the resulting trace width for a characteristic impedance of 50Ω was 0.35mm. This was the value used for the PCB design, as it takes into account the copper trace height. A useful consideration when creating RF PCBs, is that the trace impedance can easily be manipulated by changing the distance between the trace and the ground plane. This information is useful when connecting a trace with the target impedance, to a larger trace/pad with a lower impedance. By making a cut out in the ground plane (so long as there is another ground plane below), the impedance can be increased to match the target impedance.

Referring to figure 4.3, all of the four layers contain a ground plane. This was done for simplicity. The most important ground layer is on the internal layer 1 (yellow), which is located directly beneath the top layer which is where the transmission lines are located. Running along the certain edges of the board, as well as along the micro strip, are fence vias which connect all four ground planes together. The bridging between ground planes creates a vertical ground plane. This vertical ground plane helps with isolation

between transmission lines. The vias in the fence should be spaced at a distance $d \leq \frac{\lambda}{20}$. At 2.4GHz, the vias should be spaced every $\frac{0.125m}{20} = 6.25mm$. As a rule of thumb vias should be located at a distance of 4 * h from the micro strip, where h is the height of the substrate between layers. Since the fence is really intended to be between the top layer and inner layer 1, the height of the substrate between layers is 0.17mm. Thus, fence vias should be placed 0.68mm away from the transmission line. If the vias are too close to the micro strip, a waveguide could be created, which is likely to cause problems.

The distance of an object detected by the radar is proportional to the difference in frequency of the transmitted and received signals. Distance and frequency are related through time, specifically the sweep period of the ramp input. This is demonstrated in figure 4.5. Figure 4.5 shows the ramp input as a saw tooth. The actual ramp input to the VCO generated by a monolithic function generator circuit based on the XR2206 circuit is a triangular wave, however the same principle applies.

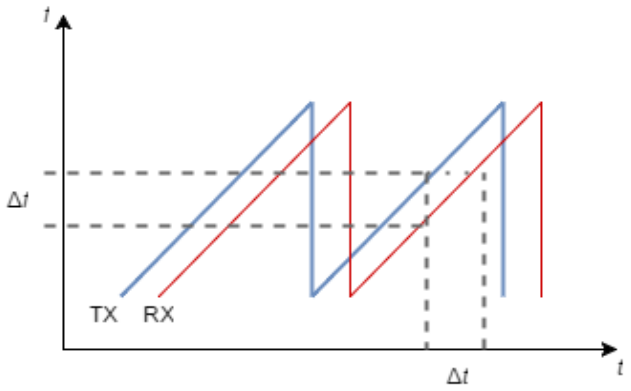


Figure 4.5: frequency vs time for TX and RX signals

Equation 4.2 displays how range, beat frequency, velocity, sweep period and bandwidth are related.

Equation 4.2:

$$R = \frac{c(Ts) * f_b}{2 * (f1 - f0)}$$

Where:

$R = Range\ (m)$

$c = speed\ of\ light\ (3x10^8 \frac{m}{s})$

$Ts = Period\ of\ frequency\ sweep$

$f_b = Beat\ frequency\ (frequency\ difference\ from\ mixer)$

$(f1 - f0) = BW_{sweep} = The\ bandwidth\ of\ the\ frequency\ sweep$

Equation 4.2 is used to determine range based on the frequency difference from the output of the mixer (after passing through a low pass filter). The bandwidth and period of the frequency sweep will be known, meaning that once the beat frequency is acquired through DSP techniques, the range can be calculated and displayed.

Evidently, the radar has a maximum range. This maximum range is determined by a variety of factors. A large object can be detected at a further distance, as it will have a larger cross sectional area, and reflect more of the transmitted power back to the radar itself. Equation 2.3 shows the relationship

between the maximum range, the transmit power, the receiver sensitivity, antenna gain, the transmit wavelength, and target cross sectional area.

Equation 4.3:

$$Rmax = \sqrt[4]{\frac{Pt * G^2 * \lambda^2 * \sigma}{\mathrm{Pr} * (4\pi)^3}}$$

Where:

$Rmax = maximum\ range\ (m)$

$Pt = Transmit\ power$

$Pr = minimum\ detectable\ receive\ power$

$G = antenna\ gain$

$\lambda = transmit\ wavelength$

$\sigma = target\ cross\ section\ (assume\ 1m^2)$

The purpose of the radar equation (equation 4.3) is to display the relationship between several key variables in radar design. The most critical relationship displayed here, is the relationship Pt,Pr and Rmax. The Pr is inversely proportional to Rmax to the fourth power. This indicates that even a small increase in range will significantly impact the receive power.

The sensitivity of the low noise amplifier (HMC667LP2) is not specified in the documentation. Therefore, the minimum detectable receive power will have to be assumed. Assuming Pr = -70dBm (0.1nW), Pt = 20dBm (100mW), G = 8.5dBi (7.08) and λ = 12.5cm, Rmax can be calculated using equation 4.3:

$$Rmax = \sqrt[4]{\frac{Pt * G^2 * \lambda^2 * \sigma}{\mathrm{Pr} * (4\pi)^3}} = \sqrt[4]{\frac{100mW * 7.08^2 * 0.125^2 m * 1m^2}{0.1nW\ (4\pi)^3}}$$

$$Rmax = 25\ m$$

The resulting maximum range is 25m. The sensitivity of the low noise amplifier may be quite different than what was assumed, and this maximum range value is only an estimation. In addition, similar experiments have been done (MIT radar project, for instance) with similar specifications that have resulted in radars with much higher range (~100m).

Furthermore, it is important to note that the actual power output of the radar is currently closer to -8 dBm (158uW). Using equation 4.3, the practical maximum range can be determined:

$$Rmax = \sqrt[4]{\frac{158uW * 7.08^2 * 0.125^2 m * 1m^2}{0.1nW\ (4\pi)^3}}$$

$$Rmax = 4.9m$$

Note: It would appear that the output radar has degraded with use, as initially the output power was approximately -2dBm. The suspected cause for this degradation is that the power amplifier is being driven too hard, and has been pushed into saturation for too long.

The analog output of the radar does not contain a single frequency component, as each object reflecting power back to the radar will produce a different harmonic at a frequency proportional to the object's distance. The resulting analog signal thus contains many different harmonics.

Because the only tests done on the radar were performed in a lab cluttered with objects that will absolutely reflect electromagnetic waves, it becomes quite difficult to pin point exact ranges of objects. That being said, when a large object enters the field of 'view' of the radar, the presence of either a phase change or additional harmonic can be distinctly seen. A large object consists of a human body, nearly anything metal, or decently large pieces of plastic (sometimes even a pencil at close range can be detected).

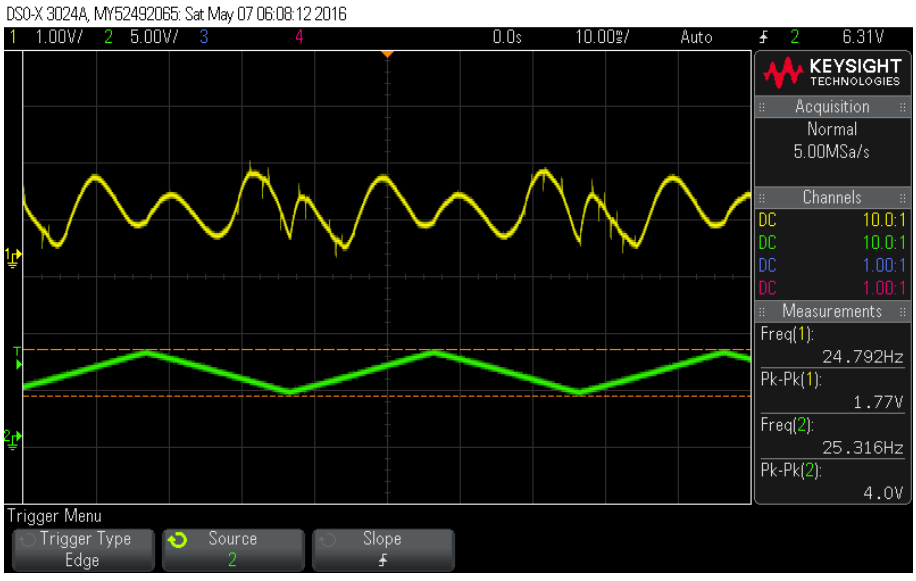
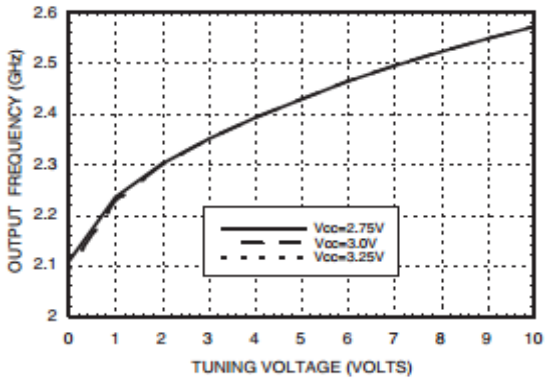


Figure 4.9: Amplified radar output (ch1) compared to ramp input (ch2).

Figure 4.9 Displays a comparison between the output of the amplification stage (discussed in Section 5.0), and the ramp/triangular wave input to the radar system. In this capture, there are several frequency components present. The most prominent harmonic appears to be 24.79 Hz. A ~25 Hz component was always present when measuring. Whether this is due to an object generating this component (as aforementioned, the test area was cluttered with so many objects reflection with the first few meters of the signal exiting the transmission antenna was guaranteed).

The ramp waveform shown in figure 4.9 has an amplitude of 4.0Vpp, and an offset of approximately 5.5V. From the VCO frequency vs. tuning voltage curve found in the HMC385LP4 datasheet, the corresponding frequency range can be seen. The intended bandwidth was 100 MHz, 2.4GHz to 2.5GHz. Ideally, a 3Vpp signal with a 5.5VDC offset would be used to generate these sweep. Indeed, this calibration was possible on the breadboard, however there appears to be a difference with the function generator circuit on the PCB (perhaps due to different component tolerances – the same components were not transferred over to the PCB). Nevertheless, a similar frequency range could be achieved (albeit, with a slightly larger bandwidth).



The frequency vs tuning voltage curve can be seen in figure 4.10. Seeing as the waveform is centered at ~5.5V, the deviation of 2V on either side results in a voltage range from 3.5V to 7.5V. This puts the theoretical frequency range at ~2.37 GHz to ~2.5 GHz. The resulting bandwidth will be slightly larger than 100MHz. For this application, this is not a concern, as this is only slightly out of the specified operating range of the power amplifier (2.4 GHz – 2.5).

Figure 4.10: Frequency vs. Tuning Voltage at T=25 deg. Celsius.

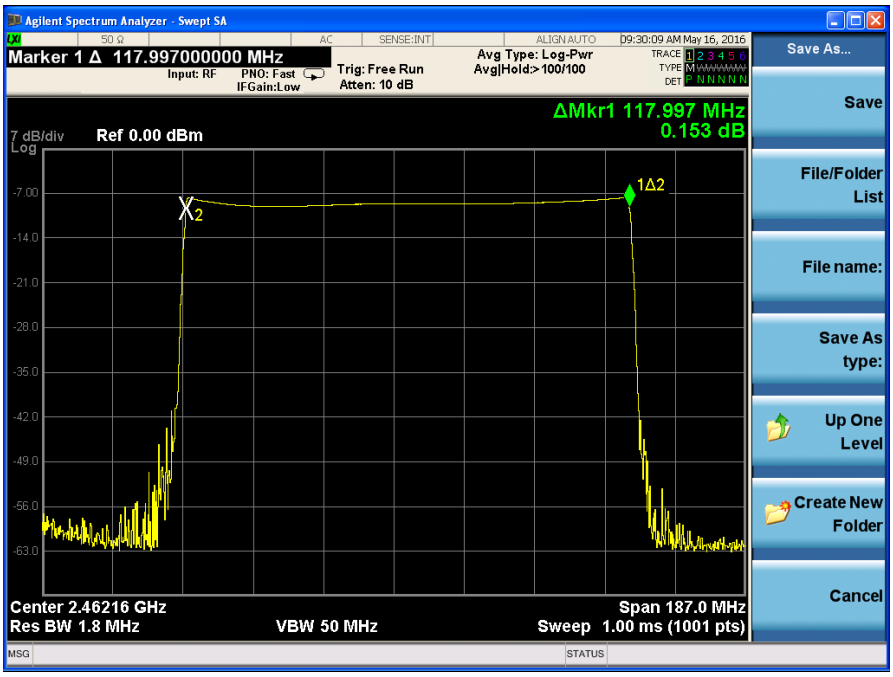


Figure 4.11: Measurement of the bandwidth of the transmitted signal.

The measured bandwidth of the transmitted wave was 118MHz. The deviation in power is actually quite minimal over the band. The power level of the transmitted waveform was -8dBm. This is significantly lower than expected. The reason for the lower power is possibly due to an issue with the power amplifier. Perhaps an important pin was somehow left unsoldered, or the amplifier was damaged somehow. Ultimately, the actual function of the radar is correct, and the lower power just means that the range of the radar is lower than expected. Furthermore, trying to find and fix this issue would be incredibly difficult as the components used in the design of the radar system are incredibly small.

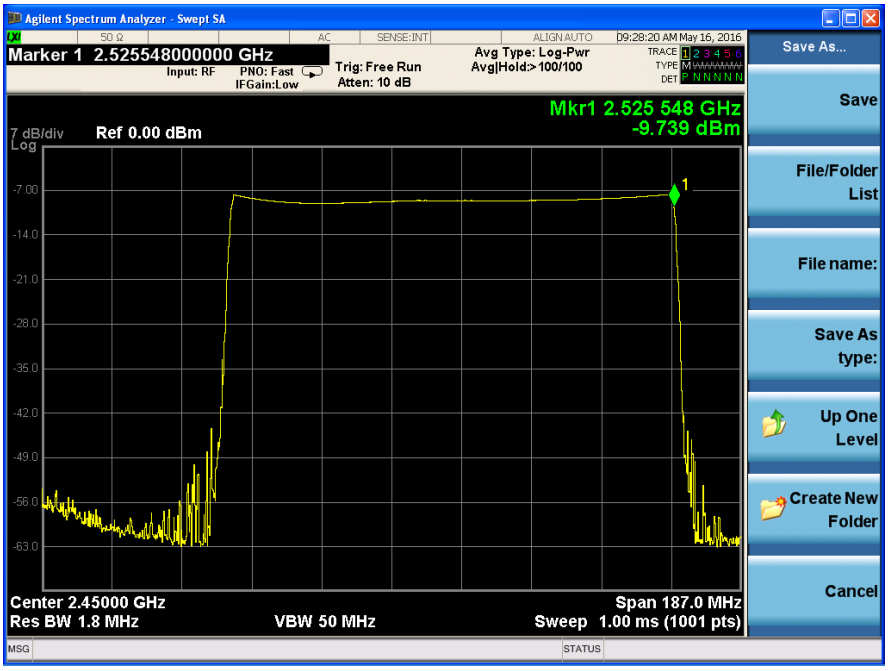


Figure 4.12: Measurement of upper frequency of the band.

The upper frequency of the band was found to be 2.525 GHz, not too far from the expected 2.5GHz based on the ramp input.

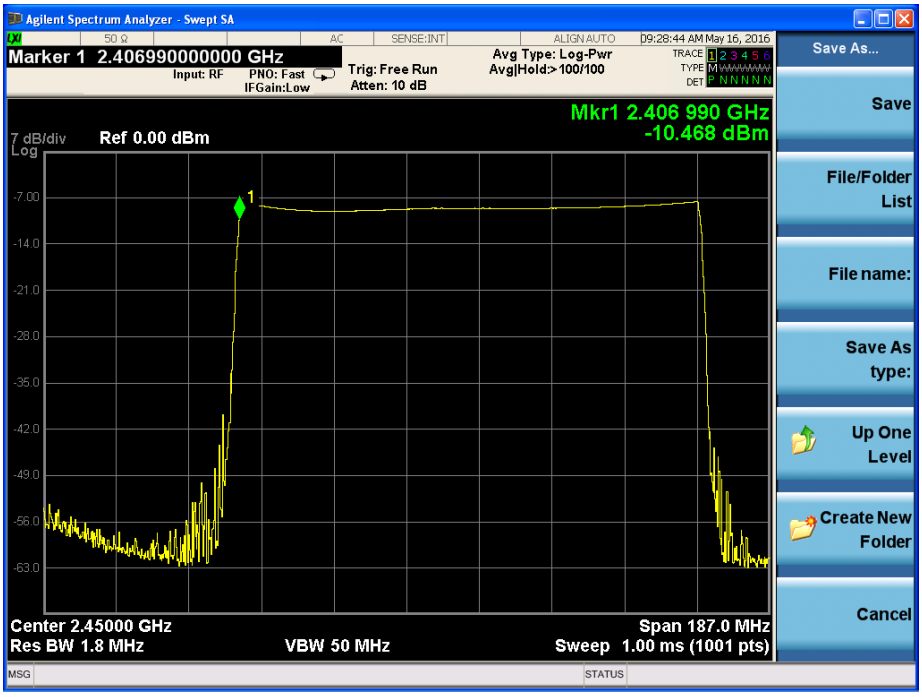


Figure 4.13: Measurement of lower frequency of the band.

The lower frequency of the band was found to be 2.406 GHz. This is different than what was expected based on the calibration curve, however it is actually beneficial as 2.4 GHz was the target anyway.

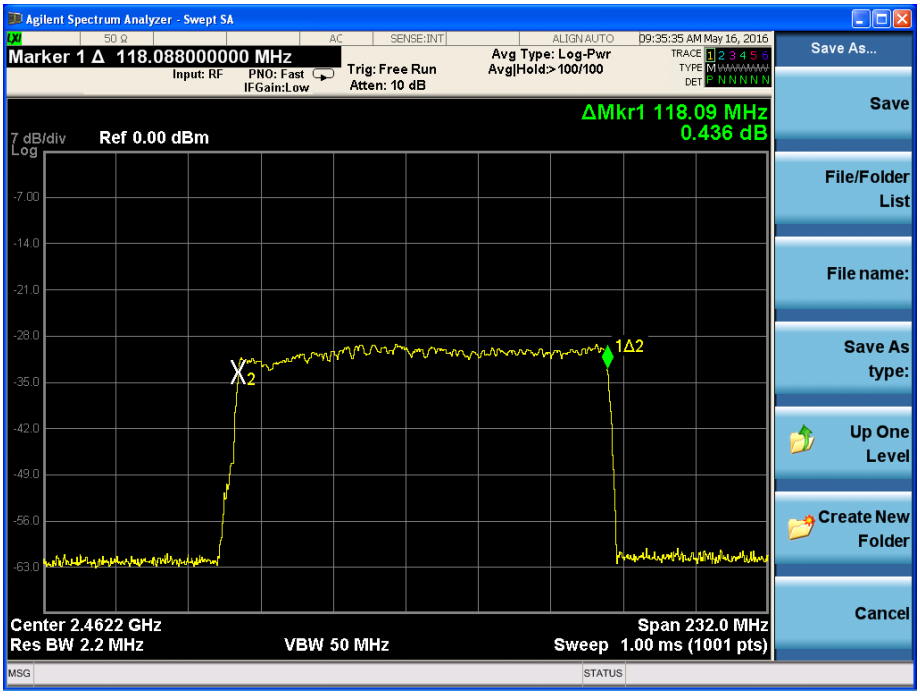


Figure 4.14: Received echo bandwidth.

Measurement of the received signal bandwidth with a metal object approximately 1 meter away from the radar. The bandwidth remains the same, however the signal power has dropped to ~-25dBm.

.These measurements show that the transmission and reception of electromagnetic waves is occurring.

4.2- Antennas:

Directive antennas are an important factor in the operation of the radar. Directive antennas are chosen, as they provide higher power in the desired direction, and furthermore the directivity is important for determining the location of an object being detected relative to the radar.

Two cantennas sizes were simulated: 540mL and 796mL. The only difference between the two cantennas is the size of the can itself. All other variables (element length, element distance from the base) remained constant.

540mL Cantenna:

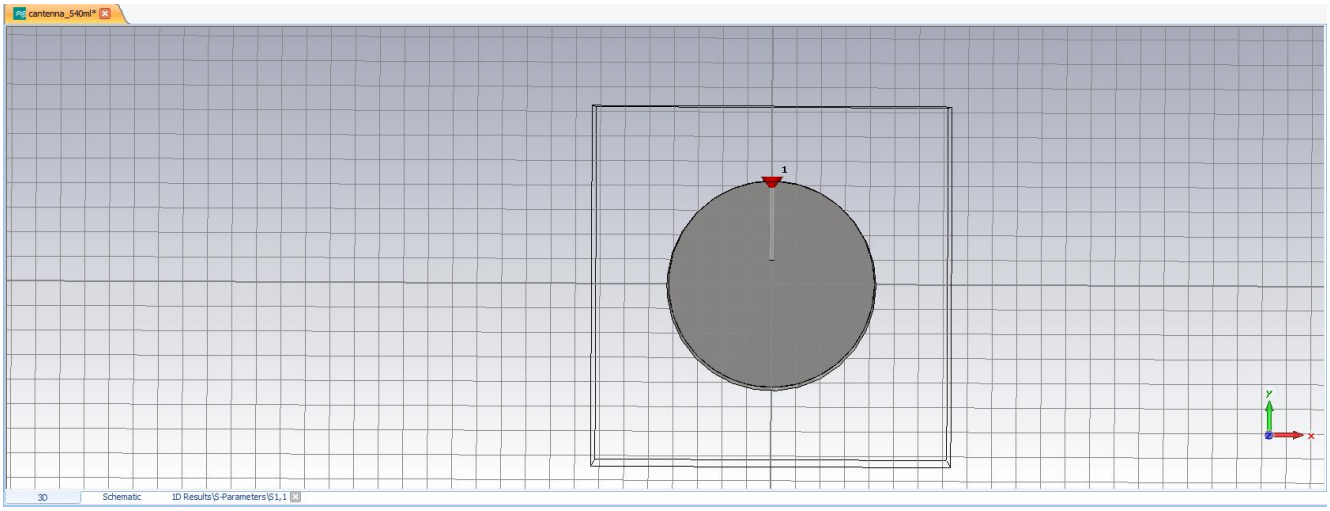


Figure 4.15: The front of the 540mL cantenna modeled in CST microwave studio.

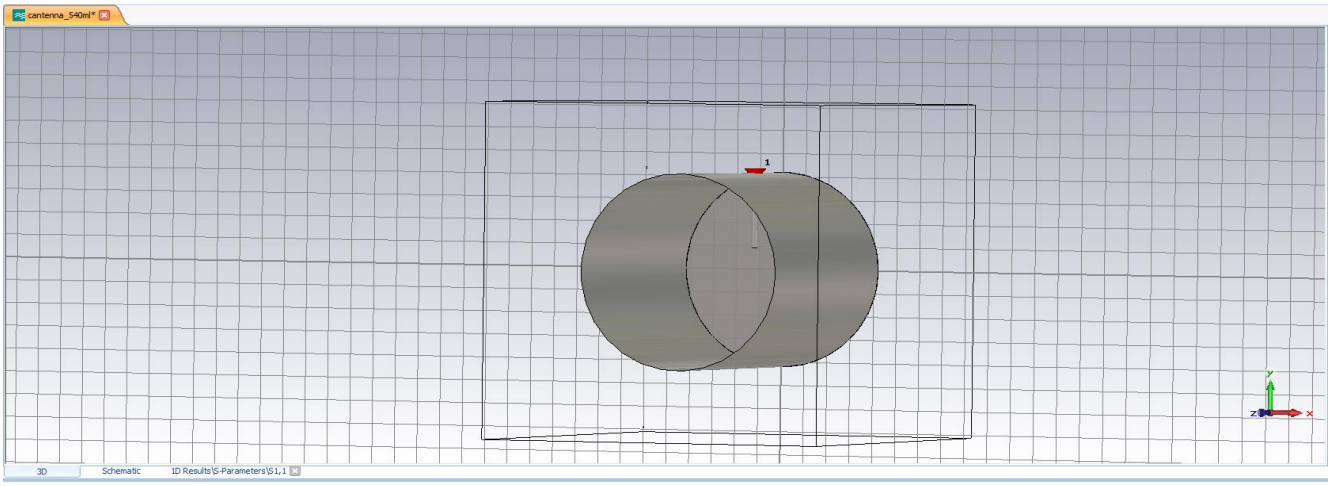


Figure 4.16: 45 degree view of the 540ml cantenna modeled in CST microwave studio.

The monopole antenna is λ/4 in length, and is spaced λ/4 from the reflector (base of the can).

The student edition of CST microwave studio has a fairly constraining limit on the number of mesh cells available for simulation. Normally, 10 mesh cells/λ is recommended, however the number of mesh

cells was dropped to 4 mesh cells/λ. This is guaranteed to produce some inaccuracies in the simulation, resulting in error in the results. In addition, certain characteristics of the can, such as the ribbing, were not modeled in the simulation. The magnitude of the effect on the farfield pattern the ribbing of the antenna causes is still to be determined.

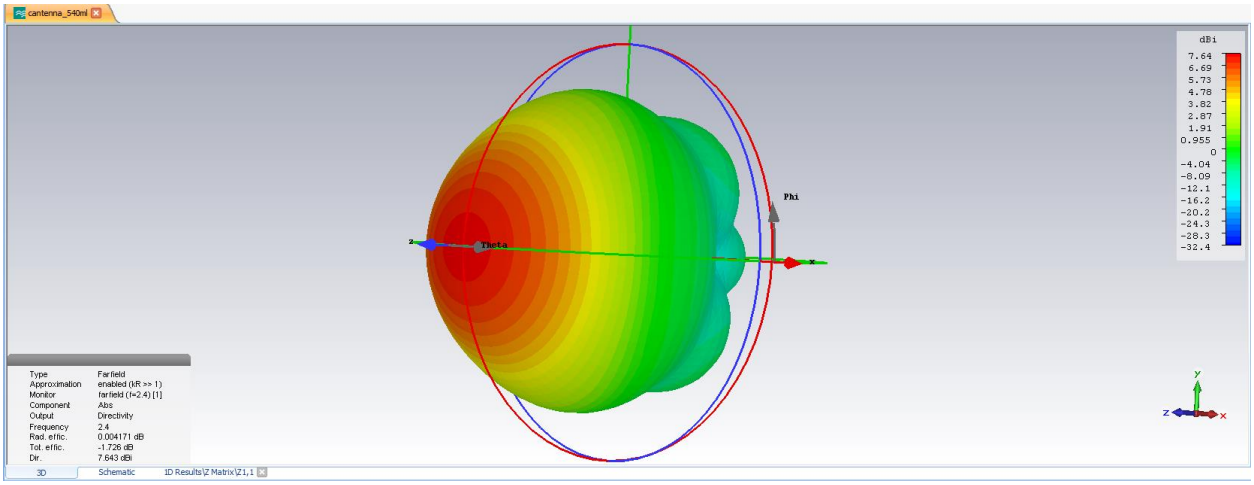


Figure 4.17: Farfield radiation pattern of the 540mL cantenna operating at 2.4GHz. Maximum gain (at 0 degrees) is 7.64dBi.

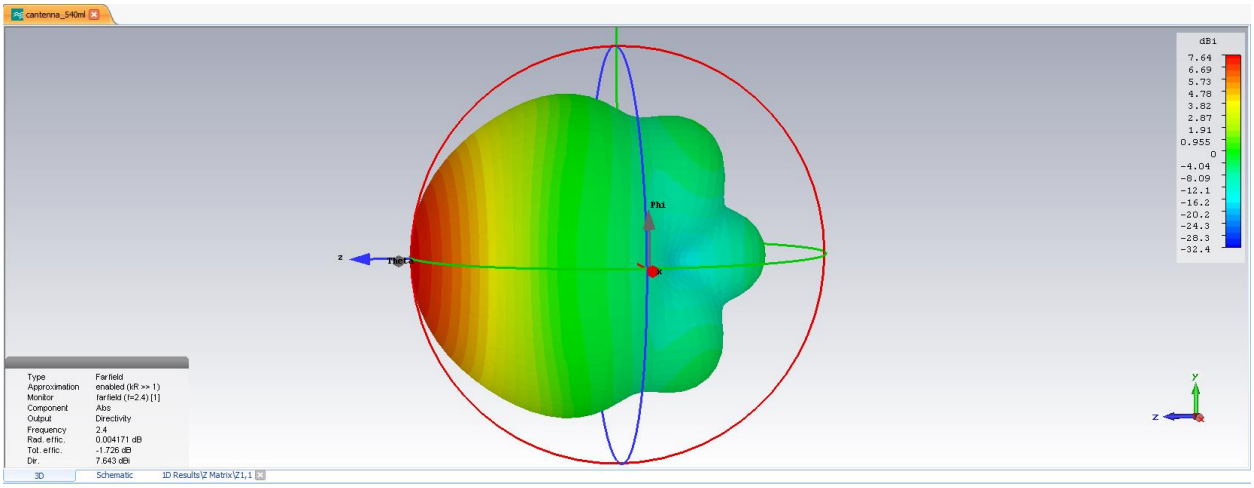


Figure 4.18: Farfield radiation pattern of the 540ml cantenna operating at 2.4GHz.

The farfield plots give an approximation of the expected farfield radiation pattern, and the expected gain at different angles.

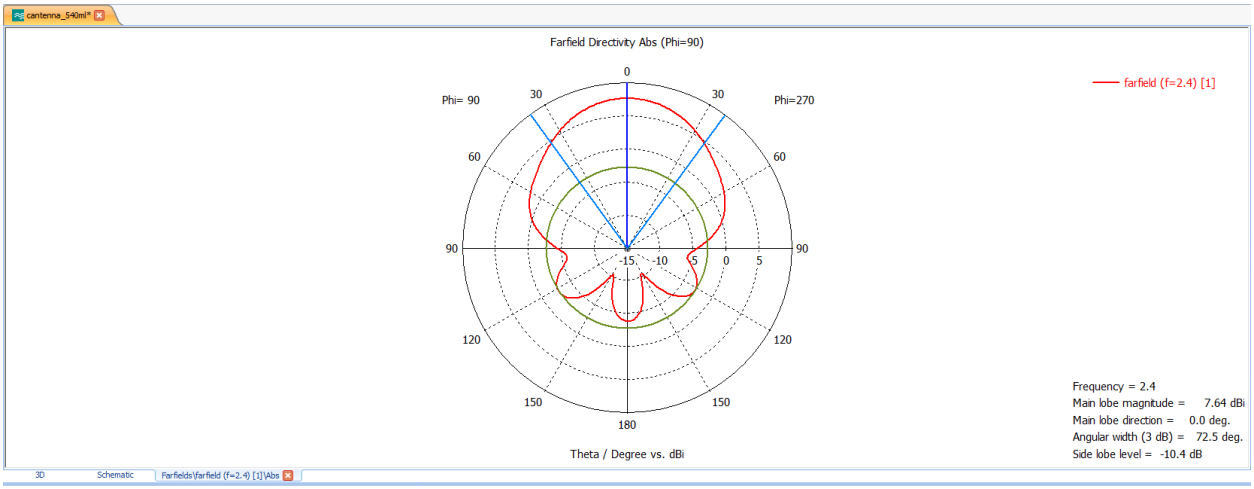


Figure 4.19: Farfield polar plot (vertical) for the 540mL cantenna.

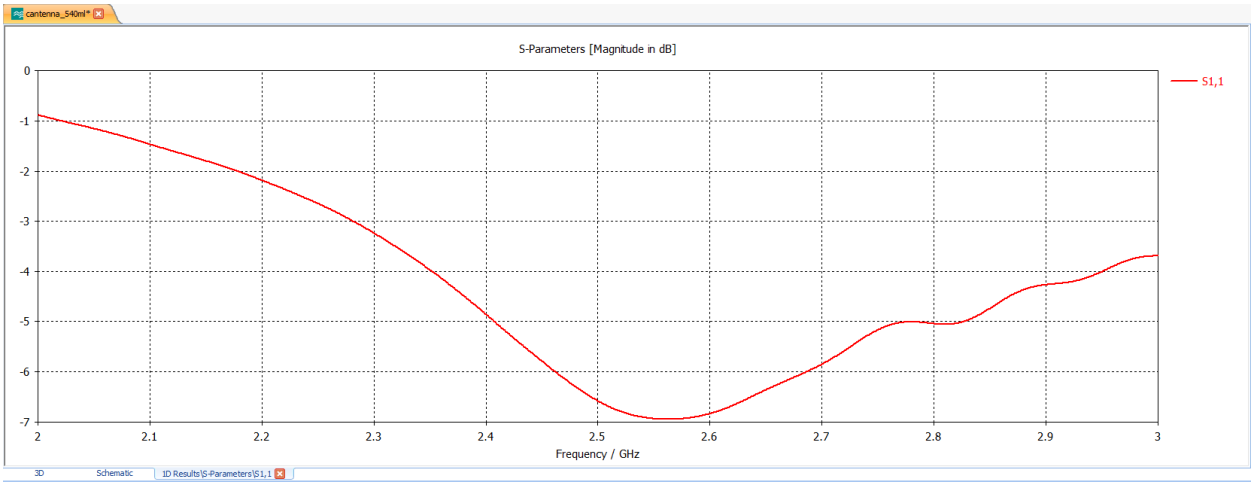


Figure 4.20: S11 parameter graph for the 540mL cantenna (-5dB at 2.4GHz).

S-parameter S11 is the reflection at the coaxial input of the antenna. The reflection ranges from -5dB at 2.4 GHz to ~-7dB at 2.5 GHz. Ultimately, this change in S11 across the sweep (2.4 to 2.5 GHz) is not an issue.

The 540mL cantenna simulations show that it has 7.64dBi of maximum gain, and has decent directivity. The 3dB angular width (vertical) is 72.5 degrees. In addition, the S11 parameter is -5dB at 2.4GHz. This could cause issues, as -5dB of reflected power is significant. Equation 4.4 is an approximate gain calculation which can also be used to determine the gain of the antenna.

Equation 4.4:

$$G_{max} = 10\log(\left(\frac{\pi D}{\lambda}\right)^2)$$

Where:

$G_{max} = maximum\ antenna\ gain\ (dBi)$

$D = diameter\ of\ waveguide$

$\lambda = wavelength\ of\ operation$

From equation 2.4, the maximum antenna gain for the 540mL cantenna can be calculated:

$$G_{max} = 10\log\left(\left(\frac{\pi(0.084m)}{0.125m}\right)^2\right) = 6.49dBi$$

Equation 4.4 can be used to estimate the gain value, but does not include enough parameters to match the sophistication of the simulation program.

796mL Cantenna:

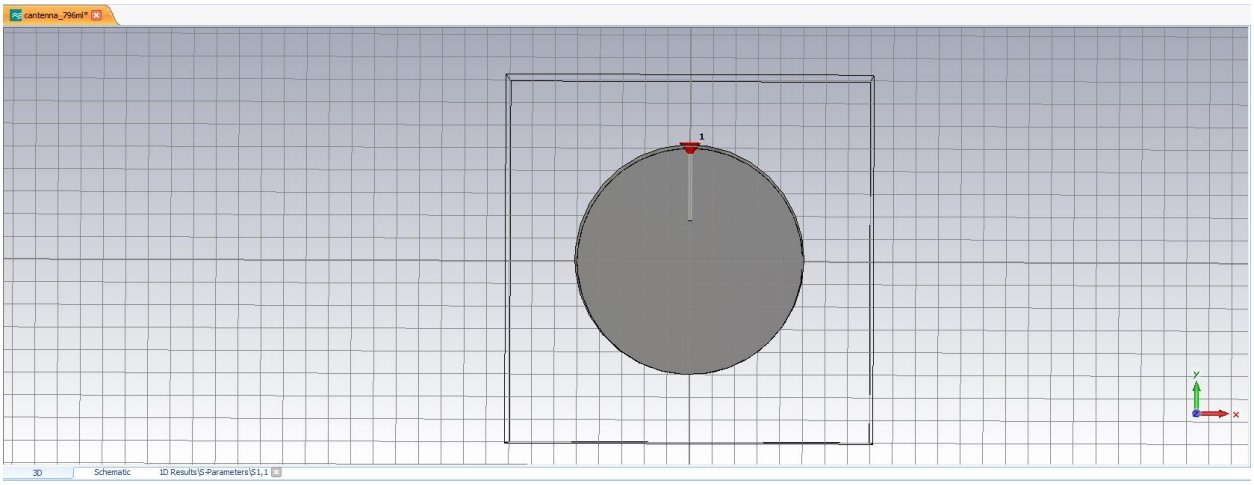


Figure 4.21: The front of the 796mL cantenna modeled in CST microwave studio.

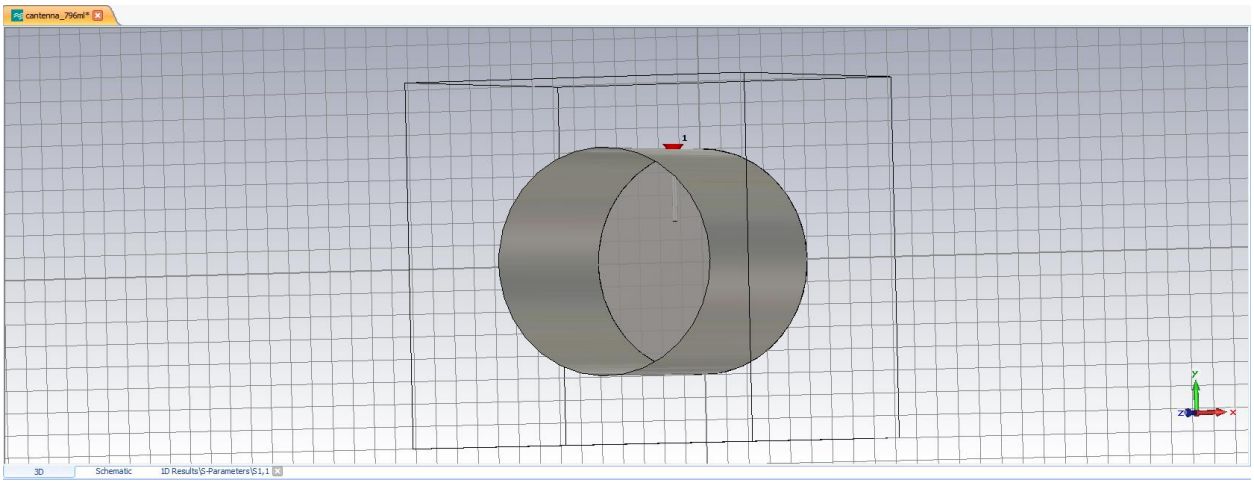


Figure 4.22: 45 degree view of the 796mL cantenna modeled in CST microwave studio.

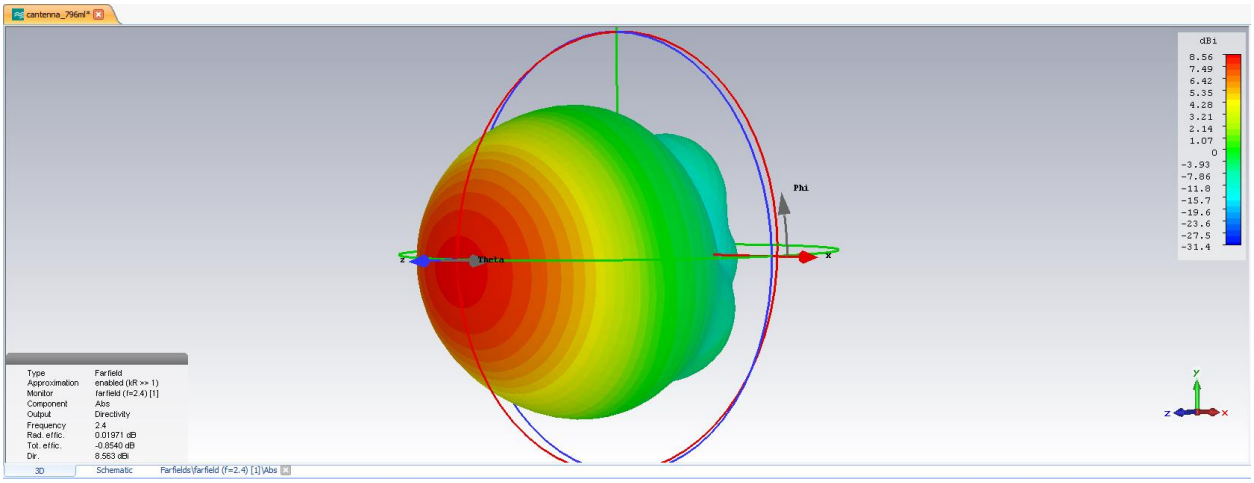


Figure 4.23: Farfield radiation pattern of the 796mL cantenna operating at 2.4GHz. Maximum gain (at 3 degrees) is 8.56dBi.

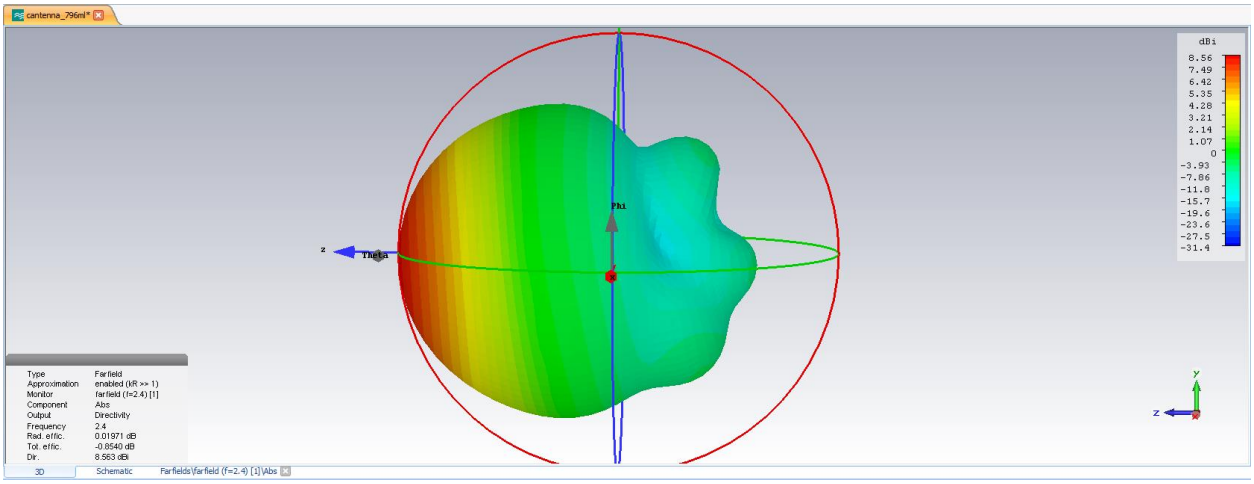


Figure 4.24: Farfield radiation pattern of the 796mL cantenna operating at 2.4GHz.

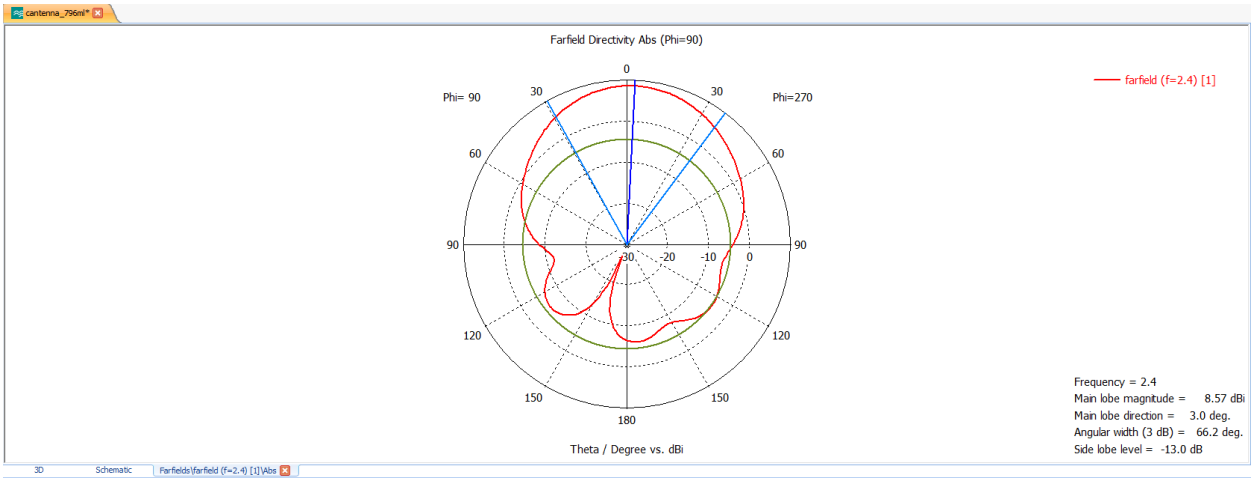


Figure 4.25: Farfield polar plot (vertical) for the 796mL cantenna.

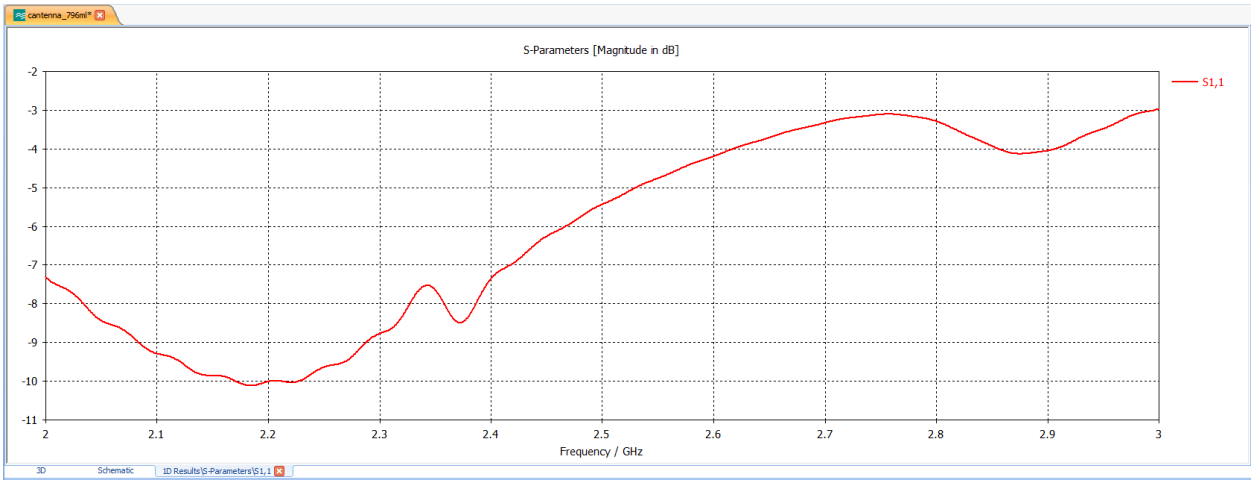


Figure 4.26: S11 parameter graph for the 796mL cantenna (-7dB at ~2.4GHz)

The 796mL cantenna appears to perform better at 2.4GHz than the 540mL cantenna. This is likely due to the size of the 796mL cantenna (larger diameter) being better suited for guiding a signal with a wavelength of 12.5cm. The 540mL cantenna might be better suited for higher frequencies. The 796mL cantenna had a maximum gain of 8.57dB, and a 3dB beam width of 66.2 deg. Therefore, it is slightly more directional (by about 6 degrees) than the 540mL cantenna. The S11 parameter is also lower for 2.4GHz, and has a magnitude of ~ -7dB. However, the 796mL cantenna has a slightly deformed radiation pattern, and the main lobe direction is actually 3 degrees off center (vertically). A cantenna with a diameter somewhere in between would likely be the ideal cantenna.

Using equation 4.4, antenna gain can be calculated for the 796mL cantenna:

$$G_{max} = 10\log\left(\left(\frac{\pi(0.1m)}{0.125m}\right)^2\right) = 8.0dBi$$

Once again, this formula (equation 4.4) produces a fairly accurate estimation of antenna gain.

Ultimately, the can that was chosen to make the cantennas was the 540mL can. This was due to the availability of two 540mL cans. The difference between the two can sizes is negligible for the purpose of this project (a prototype design). If this were a product to be manufactured, professionally designed antennas would be used. These cantennas are merely for the purpose of prototyping.

Section 5.0 – Radar Station

The purpose of the radar station module is to generate the appropriate ramp waveform to drive the VCO of the radar module, to buffer, amplify and filter the analog output of the radar, to sample this processed analog output, and to transmit the samples via wireless RF link to a nearby receiver.

5.1- Operation and Design:

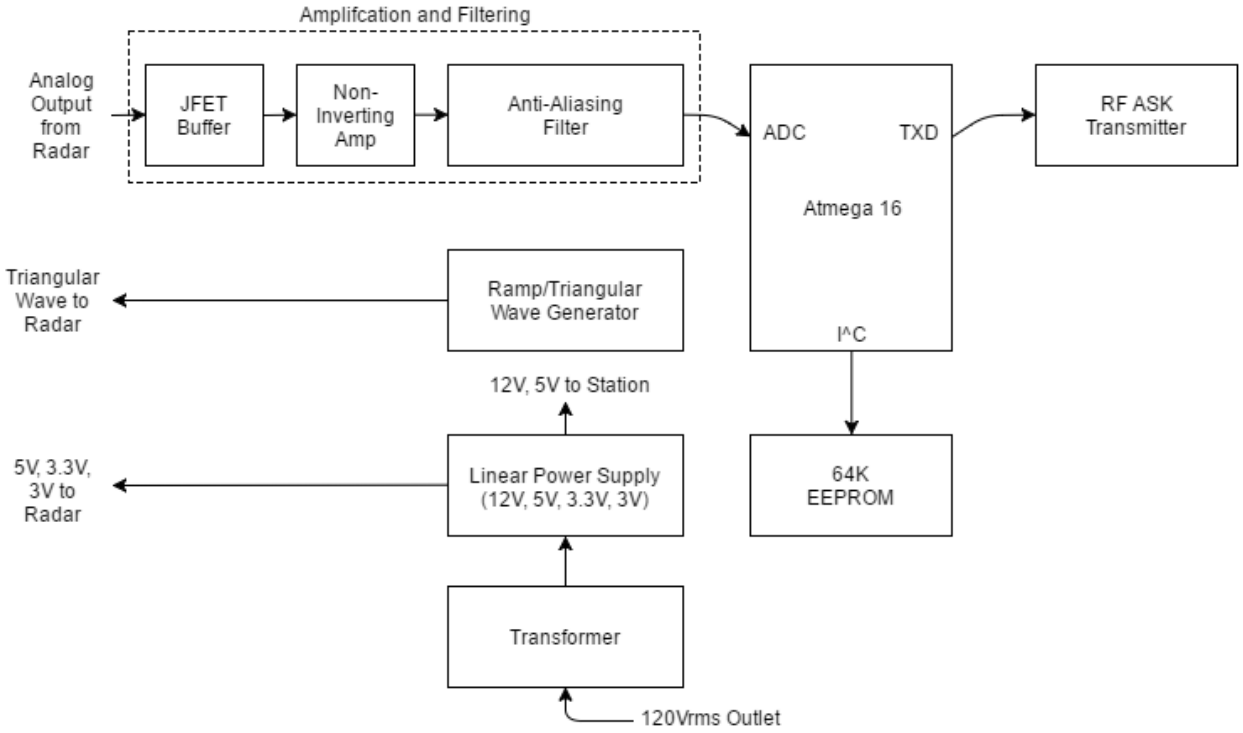


Figure 5.1: Block diagram of the radar station.

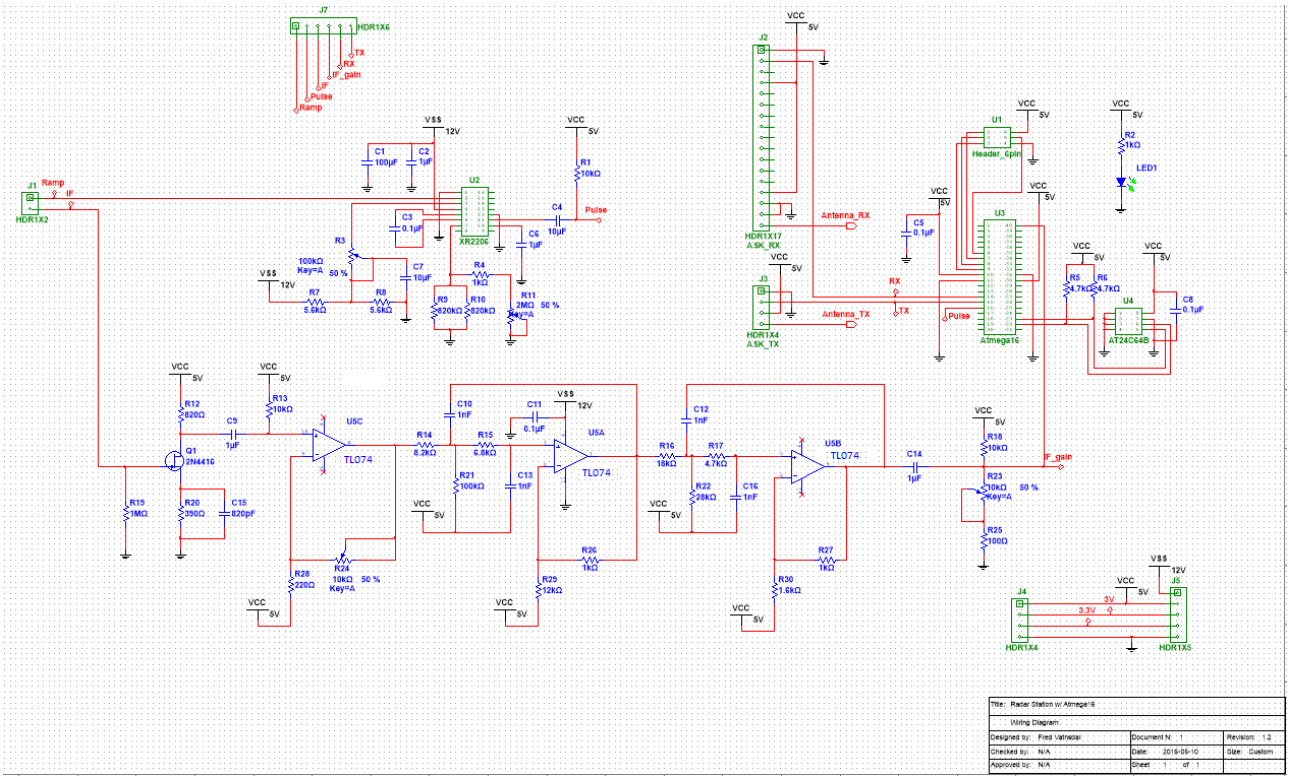


Figure 5.2: Schematic diagram of the radar station.

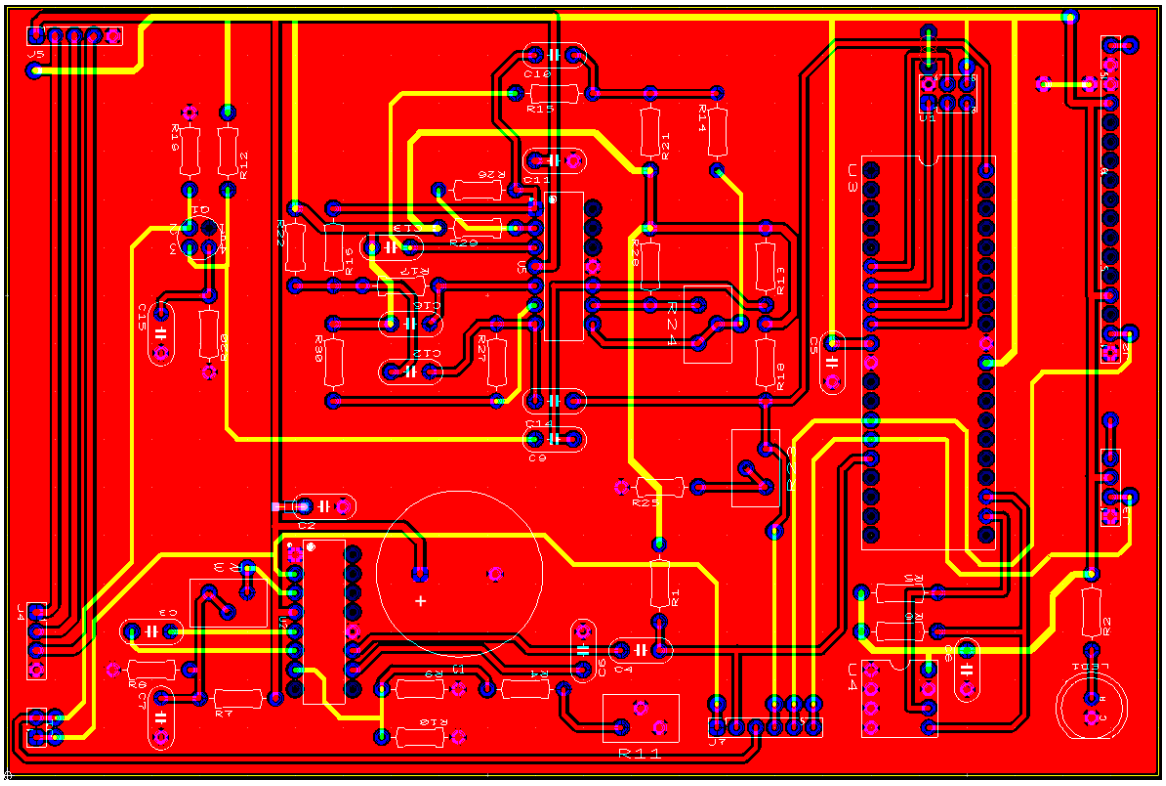


Figure 5.3: PCB design of the radar station.

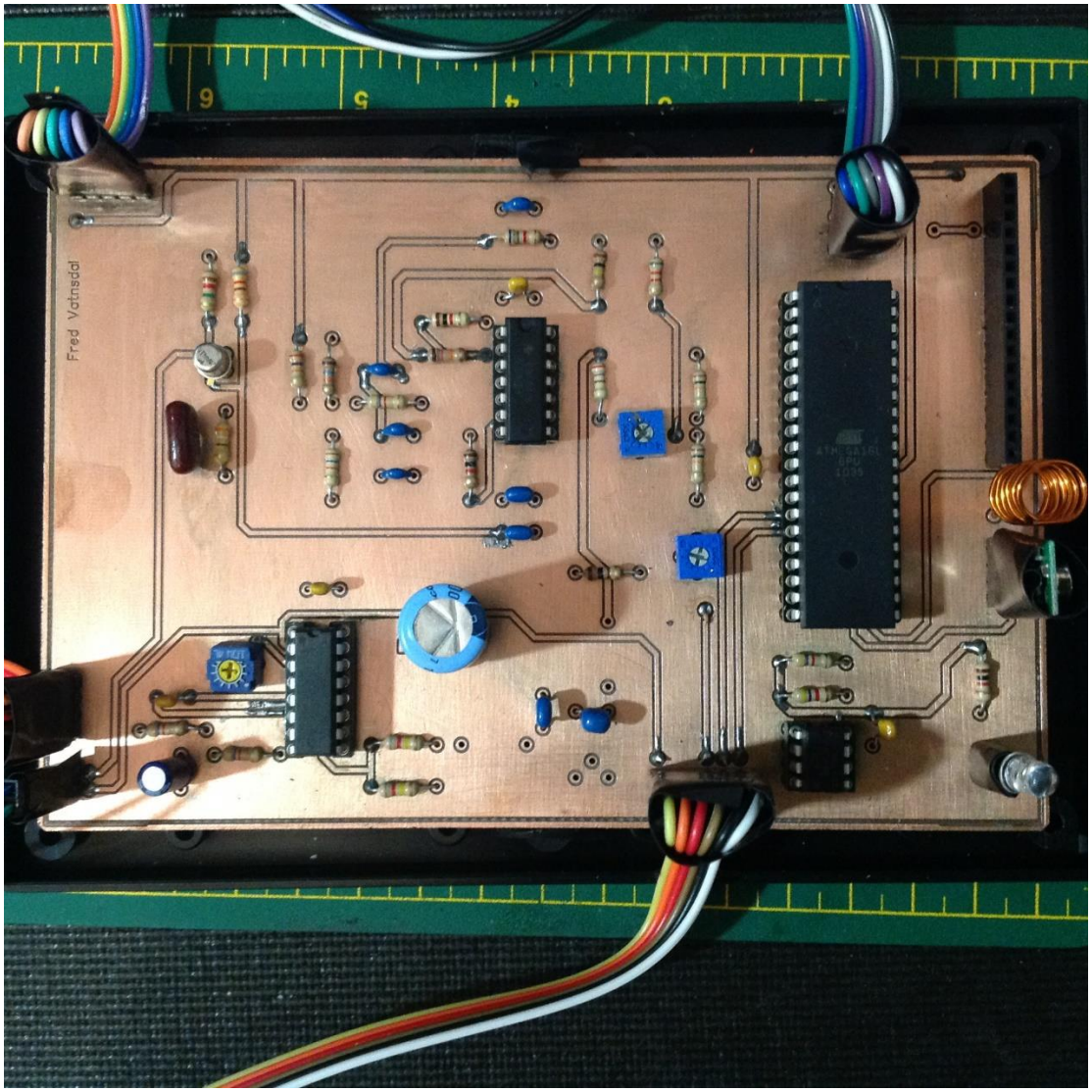


Figure 5.4: Top side of the PCB.

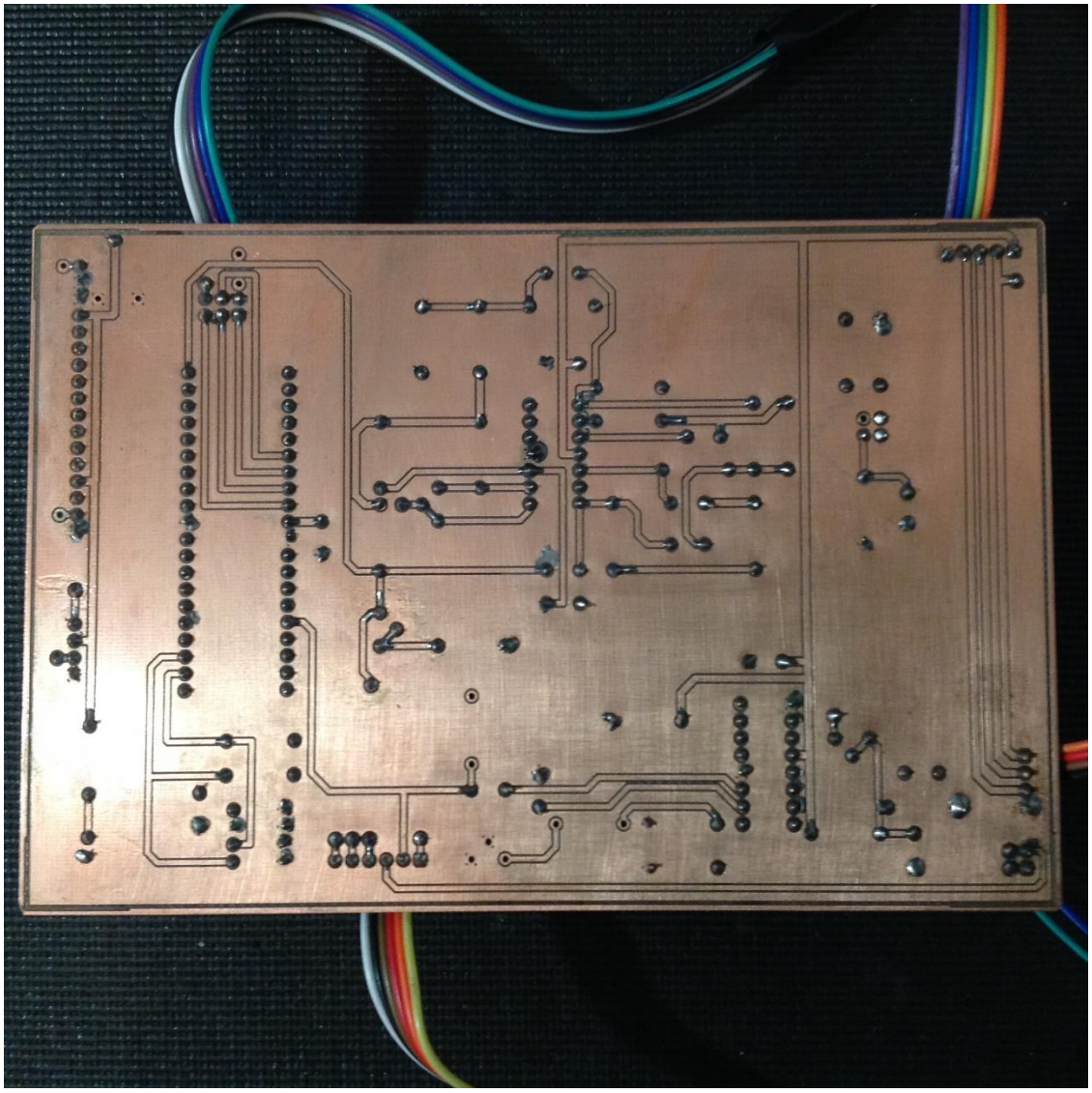


Figure 5.5: Bottom side of the PCB.

5.2-Ramp Generation:

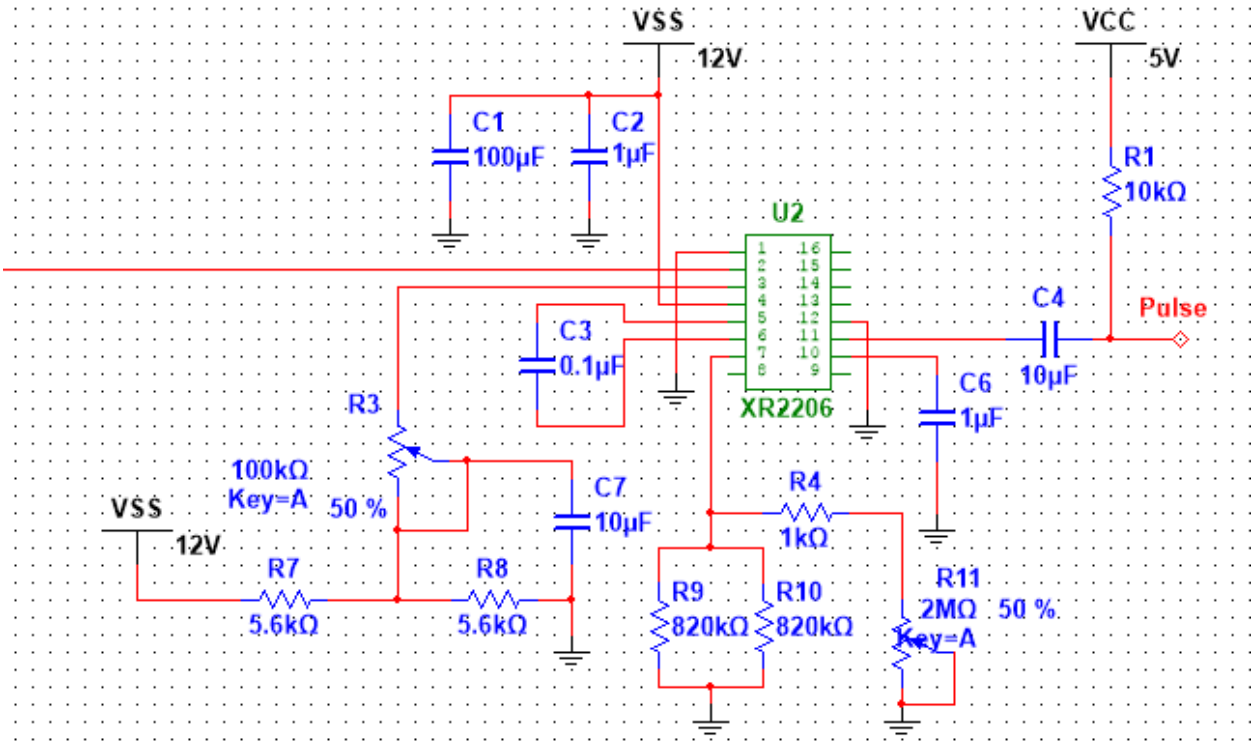


Figure 5.6: XR2206 monolithic function generator circuit used to generate the 25Hz triangular waveform.

In order to generate the waveform used to drive the VCO of the radar, the XR2206 monolithic function generator IC was used. This component is capable of keying between two frequencies, determined by C3 and the resistance at pin 7 (f1) and the resistance at pin 8 (f2). Resistors R3 and R8 were included in the design as a backup measure (in case the circuit did not behave the same as it did on the bread board). They were not soldered onto the PCB. Since only one frequency is required, only pin 7 was used. C3 and R6//R7 form an RC circuit. Thus using equation 3.1:

Equation 5.1:

$$f = \frac{1}{R6//R7(C3)} = \frac{1}{410k\Omega(0.1uF)} = 24.4Hz$$

The potentiometer R2, and the voltage divider network R4 and R5 are what determine the voltage at pin 3. This voltage determines the amplitude and offset of the output waveform. C1 and C2 are bypass capacitors used to prevent high and low frequency noise from effecting the operation of the circuit. C4 is a coupling capacitor, and R1 is a pull up resistor. The circuit operates on 12V.

In addition to the triangular wave output, a pulse output is also provided. The rising edge of the pulse indicates the start of the triangular wave cycle and can be used as a synchronization pulse. It was connected to an external interrupt pin of the Atmega16, however was not used in the final design.

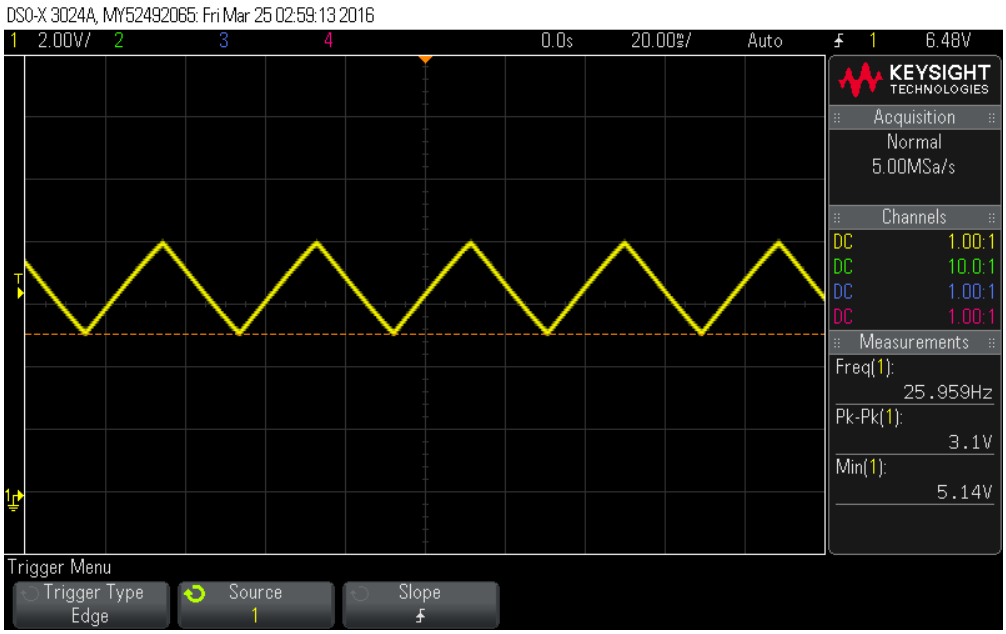


Figure 5.7: Triangular wave output on pin 2 of the XR2206 chip.

Shown in figure 5.6 is the triangular wave output. The frequency of this waveform is approximately the expected 25 Hz, and the peak to peak voltage is 3Vpp. The offset is approximately 6.6VDC, as the minimum shown is 5.14V.

5.3-Amplification and Filtering:

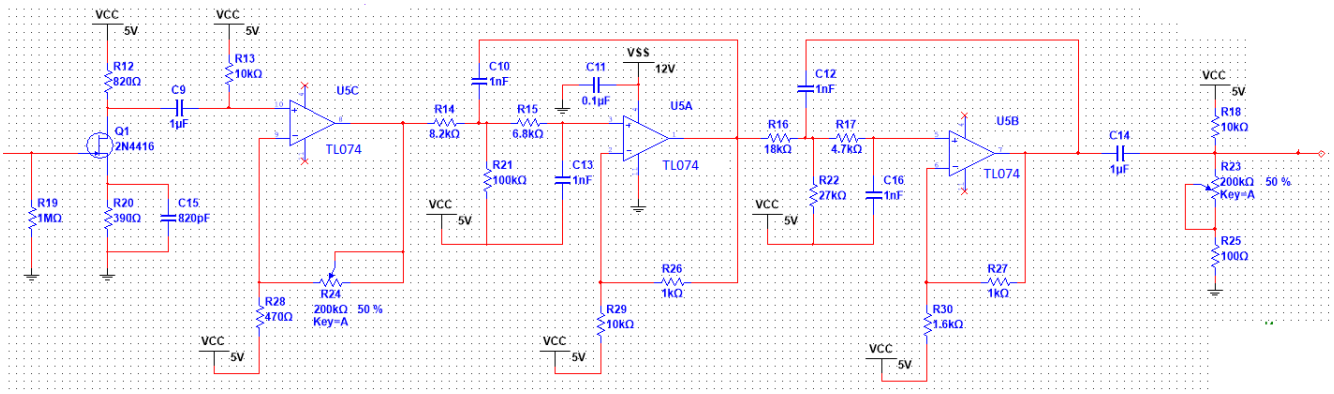


Figure 5.8: Amplification and filtering stages of the analog output of the radar.

The anti-aliasing filter design is a modification of the filter used in the MIT radar. In order to use this design, it needed to be modified in the following ways: addition of a JFET buffer stage which prevents the circuit from loading the output of the radar (the two systems cannot be interfaced without this stage), increase of the gain in the gain stage, modification of the cut-off frequency, the use of the TL074 quad op-amp chip in order to reduce cost of the design, and the addition of a level shifter at the output.

In order to sample the analog output of the radar, the signal must first be amplified and filtered. Typically, before sampling, an anti-aliasing filter is used. The purpose of this filter is to prevent signals with frequencies beyond the Nyquist limit of the sampling system (in this case, the ADC of the Atmega16) from passing through. With a 1MHz clock rate for the Atmega16, the maximum sampling frequency of the ADC on board the device is 38.5 kHz. The Nyquist limit is therefore at 19.2 kHz. Since the design shown in figure 5.7 has a cutoff of approximately 15 kHz, this is acceptable for the highest sample rate for the desired clock frequency of the microcontroller. In reality, the sample rate is much lower, and the radar is unable to detect an object that would produce a frequency component above or near 15 kHz, as the object would be out of the radar's range. This filter also serves the purpose of eliminating any high frequency components that are produced by the radar's mixer. This is quite important, as these components could skew results if sampled (although the frequency of these components is much higher than that of the sample rate, they could still effect the information signal).

The TL074 low noise quad OP-AMP chip was used. The supply rails were 12V and GND, with 5V acting as a reference point. Alternatively, +/-5V could be used to operate the circuit. It made more sense to use 12V, as the XR2206 chip already uses this voltage, and another regulator would not be required. The output of the circuit will be given a DC offset anyway, as the ADC onboard the Atmega16 should not be used to sample negative voltages, as it can damage the device.

The JFET buffer stage was a necessary addition to the circuit, as the OP-AMPs on the TL074 seemed to cause a loading effect on the radar's output. Even buffer stages created using OP-AMPs, with high input impedances, caused this effect. The JFET buffer was the final solution, and now there is no loading effect on the radar due to the high input impedance to the JFET stage. Practically, the gain of the JFET stage was found to be quite minimal (AV = ~1 to ~1.5). The input impedance of the JFET is set with R16, a 1.2MΩ resistor.

The next stage in the design is the amplification stage. This is stage was built using a non-inverting amplifier. R21 ranges from 0-200kΩ, and R25 is 270Ω. Equation 3.2 can be used to determine the gain range of this amplifier:

Equation 5.2:

$$Av = \frac{R24}{R28} + 1$$

Thus:

$$Av_{min} = \frac{0\Omega}{270\Omega} + 1 = 1$$

$$Av_{max} = \frac{200k\Omega}{270\Omega} + 1 = 741$$

Av can range from 1 to 741, however this maximum gain is unrealistic, as even low voltage signals will surpass the voltage of the rails with such high gain, resulting in saturation of the amplifier. For the target AV of 10, the trim pot would need to be set to 2.430kΩ. The reason such a large trim pot was chosen, was to have the option of having higher gain.

The filter stages are active, and thus have gains of their own. The gain each of the filter stages can also be calculated using equation 5.2:

$$Av_{filter1} = \frac{R26}{R29} = \frac{1k\Omega}{10k\Omega} + 1 = 1.1$$

$$Av_{filter2} = \frac{R27}{R30} = \frac{1k\Omega}{1.6k\Omega} + 1 = 1.6$$

The gain of the filter stages are near unity. The small amounts of gain provided make up for losses in the circuit. The reason the gain is not higher, is because there is a dedicated gain stage in this circuit which is capable of more than enough gain for what is needed. If the gain of the filters were too high, this would likely result in clipping of the signal, as the gain of the entire system would be extreme.

If Av of the gain stage is set to the reasonable 10, then the gain of this entire filter can be calculated:

$$Av = Av_{JFET} * Av_{Amp} * Av_{filter1} * Av_{filter2}$$

$$Av = 1 * 10 * 1.1 * 1.6 = 17.6$$

Ultimately, the resulting Av is equal to 17.6.

Thus, for the 100mVpp (Vin) signal output from the radar, the resulting signal will have an amplitude (Vout) of:

$$Vout = Vin * Av = 100mVpp * 17.6 = 1.76\,Vpp$$

This is perfect for sampling, as it fits within the 0 to 5V range of the ADC, but is large enough to be distinguishable when sampled.

The filtering stage is composed of two 2nd order active low pass filters, creating a 4th order low pass filter with an expected roll off rate of 80dB/decade. The first filter has the highest cutoff frequency of the two, as the second filter will steepen the roll off of the overall filter. Since this is an anti-aliasing filter, the roll off rate should be quick enough to allow the maximum amount of frequencies under the Nyquist limit to pass, but sharply cut out any frequencies higher. In order to determine the cutoff frequency of a 2nd order low pass filter, equation 5.3 can be used:

Equation 5.3:

$$fc = \frac{1}{2\pi\sqrt{R1 * C1 * R2 * C2}}$$

Using equation 5.3, the cutoff frequency of the first filter stage, fc1, and the cutoff frequency of the second filter stage, fc2, can be found.

$$fc1 = \frac{1}{2\pi\sqrt{8.2k\Omega * 1nF * 6.8k\Omega * 1nF}}$$

$$fc1 = 21.3\ kHz$$

$$fc2 = \frac{1}{2\pi\sqrt{18k\Omega * 1nF * 4.7k\Omega * 1nF}}$$

$$fc2 = 17.3\ kHz$$

Ultimately, this results in a cutoff frequency of near ~20 kHz, allowing the low frequency components that are used for range calculations to be pass through the filter and into the sampling stage. Note, these calculations are approximations of the expected cut-off frequency, as equation 5.3 does not take into consideration R21 and R22, which have an effect on the cut-off frequency. R21 and R22 are used to further tweak and ultimately reduce the cut-off frequency. These resistors actually move the cut-off frequency to around 15 kHz.

The output of the amplification and filtering stage is passed through the coupling capacitor C13 to remove any DC offset that may be present in the output. Then the level shifter composed of R15, R20 and R22 is used to apply a (variable) DC offset to the signal. It is important to have control over the DC offset, as the ADC onboard the Atmega16 used to sample the signal, requires a signal ranging from 0V to 5V. Thus if the gain of the signal can adjusted, having control over the DC offset is also required in order to keep the signal within the 0V to 5V range.

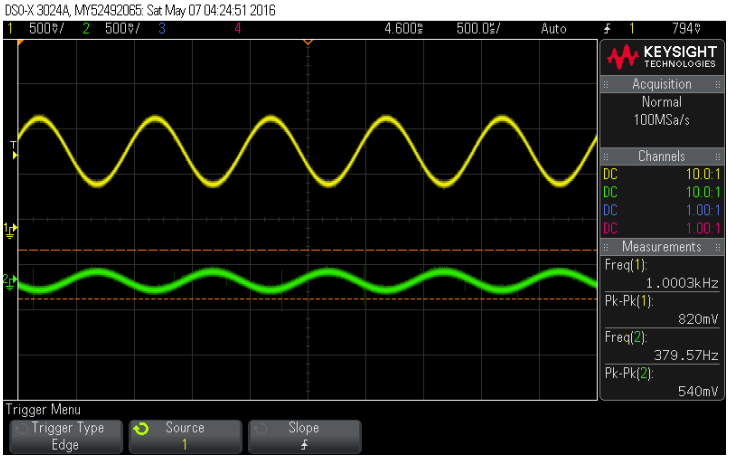


Figure 5.9: Output (ch1) vs input(Ch2) of the amplification and filtering system shown in figure 5.7 with a 1kHz sinewave input.

With a 1 kHz sinewave input, the signal is beneath the cutoff frequency and is therefore not being attenuated.

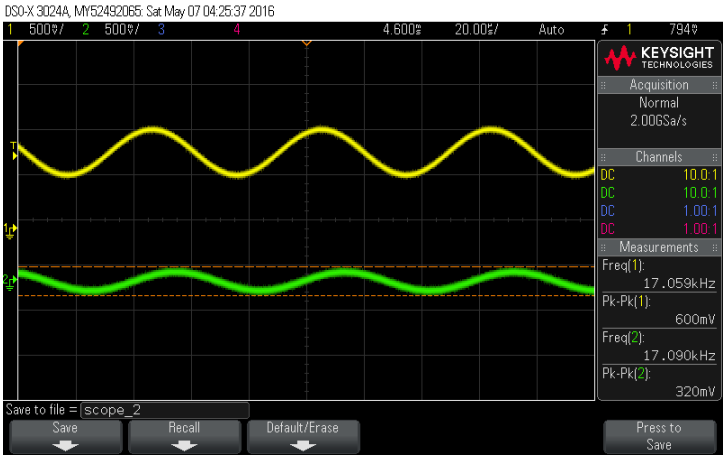


Figure 5.10: Output (ch1) vs input (Ch2) of the amplification and filtering system shown in figure 5.7 with a 17 kHz sinewave input.

A 17 kHz sinewave input is quite close to the cut off frequency of the system. Compared to figure 5.8, it can be seen that the signal has begun to degrade. As frequency increases, the signal is expected to continue to be attenuated.

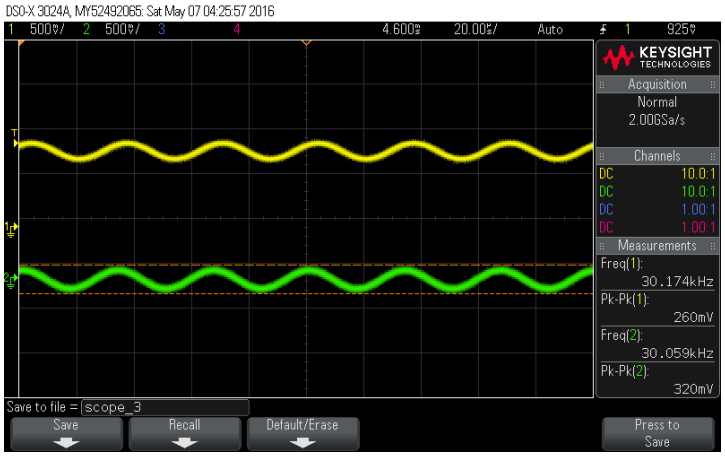


Figure 5.11: Output (ch1) vs input(Ch2) of the amplification and filtering system shown in figure 5.7 with a 30kHz sinewave input.

A 30kHz sinewave input is ~10kHz passed the expected cutoff. The signal has been attenuated significantly, and has reached an amplitude (260mVpp) lower than the input signal (320mVpp). Recall that there is a high gain stage before the filter, meaning that the attenuation of the filter is actually quite a bit more significant at this frequency than at first glance based on a comparison between input and output. Figures 5.8 – 5.10 are a demonstration of the filter's operation. The filter will be easily able to block the unwanted high frequency components generated by the radar, as these frequencies would typically be in the gigahertz.

5.4 - Sampling, Transmitting and the Microcontroller:

In order to convert the analog signal generated by the radar into data that can be processed by a modern computer, it must first be converted into a digital representation. Many microcontrollers these days have integrated ADC functions. One such microcontroller is the Atmega16. This device comes equipped with a 10 bit ADC supporting single ended and differential operational modes, selected through internal registers. For the purposes of this prototype, only the eight most significant of the ten bit resolution will be used, as that means an entire sample would be able to be sent in a UART frame, instead of breaking the sample into two pieces to be sent and reconstructed at the receiver end. Single ended mode was used in the final design, however using the differential mode was contemplated at one point, and the original amplification + filtering system included a single ended to differential mode stage.

The reference voltage of the ADC is from 0 to 5V. Since there are 10 bits sampled, the voltage resolution of the ADC can be calculated using equation 5.4:

Equation 5.4:

$$Vrez = \frac{Vrange}{2^n - 1} = \frac{5V}{2^{10} - 1} = 4.88mV$$

This voltage resolution will be used to determine the amplitude of the sample received at the digital signal processing end.

The microcontroller's roles can be broken down into two categories: sampling and transmitting. The firmware for the microcontroller was written in C. Just like any peripheral, the Atmega16's ADC is configured through the appropriate registers. The Atmega16 has a variety of operational modes, some more sophisticated than is required for this project. The ADC only has to sample on command, and is triggered manually. The configuration of the ADC is done through registers ADMUX and ADCSRA. Code Snippet 5.1 shows how the ADC can be initialized to use the AVCC as a reference, and to have a sample rate of fclk/(64 * 13clk cycles). ADMUX provides control over voltage references and left adjustment of results. Setting REFSO to 1 results in the microcontroller using the voltage at AVCC as a reference. ADCSRA is the register than controls operation of the ADC, and is where the ADC clock rate (and thus the ADC sample rate) is configured. The ADC is enabled by setting ADEN to 1, and the sample rate determined by the values in the ADPSx bits. Each combination in this series of 3 bits represents a division factor. '111', as shown in figure 3.1, is a division of the clock frequency by 64. For a clock frequency of 1MHz, this results in a sample rate of 1.2 kHz (1 MHz/(64*13)). Although a low sample rate, this is excellent for the purpose of sampling a signal in the hundreds (or less) hertz.

```
void init_adc(){
       ADMUX = (1<<REFS0);
       ADCSRA =(1<<ADEN)|(1<<ADPS2)|(1<<ADPS1);
       return;
}
```

Code Snippet 5.1

Next, the ADC must be used to sample the signal. This is done by selecting the channel of operation in ADMUX, starting the conversion using the ADSC bit in the ADCSRA register, and then waiting until the conversion is complete. A function named 'start_conversion' was created to perform this task. This function can be seen in Code Snippet 5.2. The function returns the 10 bit value gathered by the ADC in and will be used by a higher level function to gather the required number of samples.

```
int start_conversion(char channel){
	channel = channel & 0x07;
	ADMUX = ADMUX | channel;
	ADCSRA |= (1<<ADSC);
	while(ADCSRA & (1<<ADSC));
	return(ADC);
}
```

Code Snippet 5,2

A variable amount of samples can be collected, and the perfect amount of samples to collect is still an unresolved question. For the demonstration of this project, 128 samples was the chosen amount. This amount can accurately characterize a decent chunk of the time domain signal, while remaining a relatively small number of samples. The problem with large sample chunks, is that they take a huge (relatively) amount of time to gather, transmit and process. Since the DSP program is (currently) not multi-threaded, the delay from even twice the number of samples (256) is quite noticeable. Thus, the number of samples taken is 128.

128 samples are taken, and stored in an array. Now they must be transmitted. Conveniently, the Atmega16 (like the many microcontrollers) contains a USART. This will be the method of transportation for the samples. Data is transmitted serially when using USART. The USART must be configured before it can be used. This is accomplished using an 'init_usart' sub routine built for this purpose. This function can be seen in Code Snippet 5.3. First, the baud rate is set using UBRRH and UBRRL. The baud rate is determined based by the formula shown in equation 5.5.

```
void init_usart(){
	UBRRH = 0;
	UBRRL = 0x33;
	UCSRB = (1<<RXEN)|(1<<TXEN);
	UCSRC = (1<<USBS)|(1<<UPM1)|(1<<UCSZ1)|(1<<UCSZ0);
	return;
}
```

Code Snippet 5.3

Equation 5.5

$$Baud = \frac{fosc}{16(UBRR + 1)}$$

The microcontroller is operating at 1MHz, and the combination of UBRRH (0) and UBRRL (0x33). Thus the baud rate can be calculated for the UBRR setting in Code Snippet 5.3 using equation 5.4.

$$Baud = \frac{1MHz}{16(51 + 1)} = 1200baud$$

This baud rate satisfies the two criteria: it is under 4800baud (maximum baud rate of RF link) and that it is a standard baud rate. The data rate is not incredibly important, as the data processing done on the computer will take time (single thread) and data will be missed anyway. The next section of the code snippet is where the TX and RX functions of the USART are activated by setting the respective bits in the UCSRB register to 1. The parameters for the data such as parity and stop bits are configured in UCSRC. USBS defines the number of stop bits. Setting USBS to 1 indicates that there are two stop bits. Setting UPM1 to 1 and UPM0 to 0, as shown, sets even parity. The UCSZ bits define the data size. In this case, 8 bits are used.

```
void transmit_data(uint8_t samples[], int numSam){
	int i;
	for(i = 0; i < numSam; i++){
		while(!(UCSRA & (1<<UDRE)));
		UDR = samples[i];
	}

}
```

Code Snippet 5.4

Transmitting was split into two functions, a bare-bones transmission function called 'usart_transmit', and a function used specifically for transmitting the data portion of the frame called 'transmit_data'. The latter function can be seen in code snippet 5.4. The function outputs the samples in the array of samples passed to it. Before outputting, the microcontroller waits until UDRE is set. Then the byte is output using UDR. The separation of transmission into two functions, one for sending framing flags, and the other for sending the actual data samples facilitates creating custom frames in the main routine.

In case a large amount of samples needed to be taken, 64K of EEPROM to store these samples was provisioned for. The AT24C64B was chosen to fulfill this role, as it has an I^2C interface, which can be used to interface with the Atmega16. In order to use the I^C lines, pull up resistors R5 and R6 must be used. The minimum and maximum design equations are found in the data sheet. The maximum pull up resistor value can be calculated using equation 5.6

Equation 5.6:

$$Rpullup\ (max) = \frac{tR}{0.8473 * Cb}$$

Maximum value of Cb 550pF, and the normal rise time, tR, is 3000ns.

$$Rpullup(max) = \frac{3000ns}{0.8473 * 550pF} = 6437\Omega$$

The minimum pull-up resistor value can then be calculated using equation 5.7.

Equation 5.7:

$$Rpullup(min) = \frac{Vcc - Vol(\max)}{Iol}$$

The value of Vcc is 5V, the value of Vol(max) is 0.4V, and the value of Iol is 2.1mA (from the data sheet).

$$Rpullup(min) = \frac{5.0V - 0.4V}{2.1mA} = 2190\Omega$$

In order to choose a resistor value within this range, the average of the maximum and minimum pull up resistors was taken. This ended up being 4.3kΩ. 4.7kΩ was the value used for the pull up resistors in the final design. So long as the maximum and minimum requirements are met, and the pull up resistor is kept under the absolute maximum of 10kΩ, the I^2C interface will function as intended.

Ultimately, the extra memory was not needed, however having the appropriate hardware makes it a viable option if the project were to be expanded upon.

Sampling and Transmit Code for Atmega16 (written in C):

```
//Fred Vatnsdal
//FMCW Radar

#include <avr/io.h>
#include <stdio.h>

#define N_SAMPLES 128          //number of samples per frame
#define PREAM 0xAA
#define START 0x53             //start flag
#define END 0x45               //end flag
void init_adc();
void init_usart();
void sample_signal(uint8_t[]);
int start_conversion(char);
void usart_transmit(int);
uint8_t transmit_data(uint8_t[],int);



int main(void){
        init_adc();
        init_usart();
```

```
	uint8_t checksum;
	uint8_t samples[N_SAMPLES];

	while(1){
		sample_signal(samples);				//Acquire samples
		usart_transmit(PREAM);				//Send two preamble bytes
		usart_transmit(PREAM);
		usart_transmit(START);				//Send start flag
		checksum = transmit_data(samples, N_SAMPLES);	//Send samples
		usart_transmit(checksum);
		usart_transmit(END);				//Send end flag
	}
}

void init_adc(){
	ADMUX = (1<<REFS0);
	ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1); //ADPS2,ADPS1,ADPS0 (currently /128)
	return;
}

void init_usart(){
	UBRRH = 0;
	UBRRL = 0x33;
	UCSRB = (1<<RXEN)|(1<<TXEN); //enable TX and RX.
	//Synch, 8 bits, even parity, 2 stop bits.
	UCSRC = (1<<URSEL)|(1<<USBS)|(1<<UPM1)|(1<<UCSZ1)|(1<<UCSZ0);
	return;
}
```

```
void sample_signal(uint8_t samples[]){
        int i;
        int sample;
        for(i = 0; i < N_SAMPLES; i++){
                sample = start_conversion(0);           //ADC 0
                sample = sample >> 2;
                samples[i] = sample;                    //Store samples
        }
        return;
}

int start_conversion(char channel){
        channel = channel & 0x07;                       //Select channel (0-7)
        ADMUX = ADMUX | channel;                        //""
        ADCSRA |= (1<<ADSC);                            //Start conversion
        while(ADCSRA & (1<<ADSC));                      //Wait until conversion complete
        return(ADC);                                    //Return the ADC reading (10 bit)
}

void usart_transmit(int data){
        while(!(UCSRA & (1<<UDRE)));                    //Wait until UDRE is set
        UDR = data;                                     //Transmit
}

uint8_t transmit_data(uint8_t samples[], int numSam){
        int i;
        uint8_t checksum = 0;                           //Iterator
        for(i = 0; i < numSam; i++){
                checksum += samples[i];
```

```
            while(!(UCSRA & (1<<UDRE)));        //Wait until UDRE is set
            UDR = samples[i];                   //Transmit
        }
        return checksum;
}
```

5.5-RF Link (Transmitter):

Figure 5.12: TWS-BS RF Transmitter.

The telecommunications link between the transmitter and receiver is composed of an ASK RF transmitter and receiver pair. The devices are from the TWS-BS/RWS-371 RF Module Series. The devices operate wirelessly and at 315 MHz. The RF transmitter has an output power of 15 dBm (from table 5.1). The receiver's minimum sensitivity according to the data sheet is -106dBm.

Table 5.1: TWS-BS RF Transmitter Electrical Characteristics

Characteristic	Min	Type	Max	Unit
Operating Frequency (±250KHz)	314.75	315.00	315.25	MHz
Data Rate			8	Kbps
Current Consumption			8	mA
Output Power			32	mW
Operating Voltage	3		12	VDC
Operating Ambient Temperature	-20		+85	℃

Using the free space loss formula (equation 5.5), an estimation of the worst case theoretical maximum range of this device can be determined.

Equation 5.5:

$$FSPL(dB) = 10\log\left(\left(\frac{4\pi d}{\lambda}\right)^2\right)$$

The maximum FSPL can be calculated using the transmitter power and the receiver sensitivity:

$$FSPL(dB)(\max) = 14\ dBm - (-106\ dBm) = 120\ dB$$

Since λ is known to be 0.95m, distance can be calculated by solving for d in equation 3.5.

$$d = \sqrt{10^{\frac{FSPL(dB)}{10}}} * \frac{\lambda}{4\pi}$$

$$d = \sqrt{10^{\frac{120dB}{10}}} * \frac{0.95m}{4\pi} = 75.6\ km$$

Evidently, a range of 75.6 km, even in free space, is absurd for this cheap transmitter-receiver pair. Practically, the range is a few feet.

The transmitter and receiver modules both use quarter wavelength Marconi antennas. The length of these antennas was calculated using equation 5.6:

Equation 5.6:

$$\lambda = \frac{Vo}{f}$$

$$\lambda = \frac{3x10^8}{315x10^6} = 0.95m$$

A quarter of 0.95m is 0.24m. Thus the length of each antenna is approximately 0.24m. The antennas were coiled seven times.

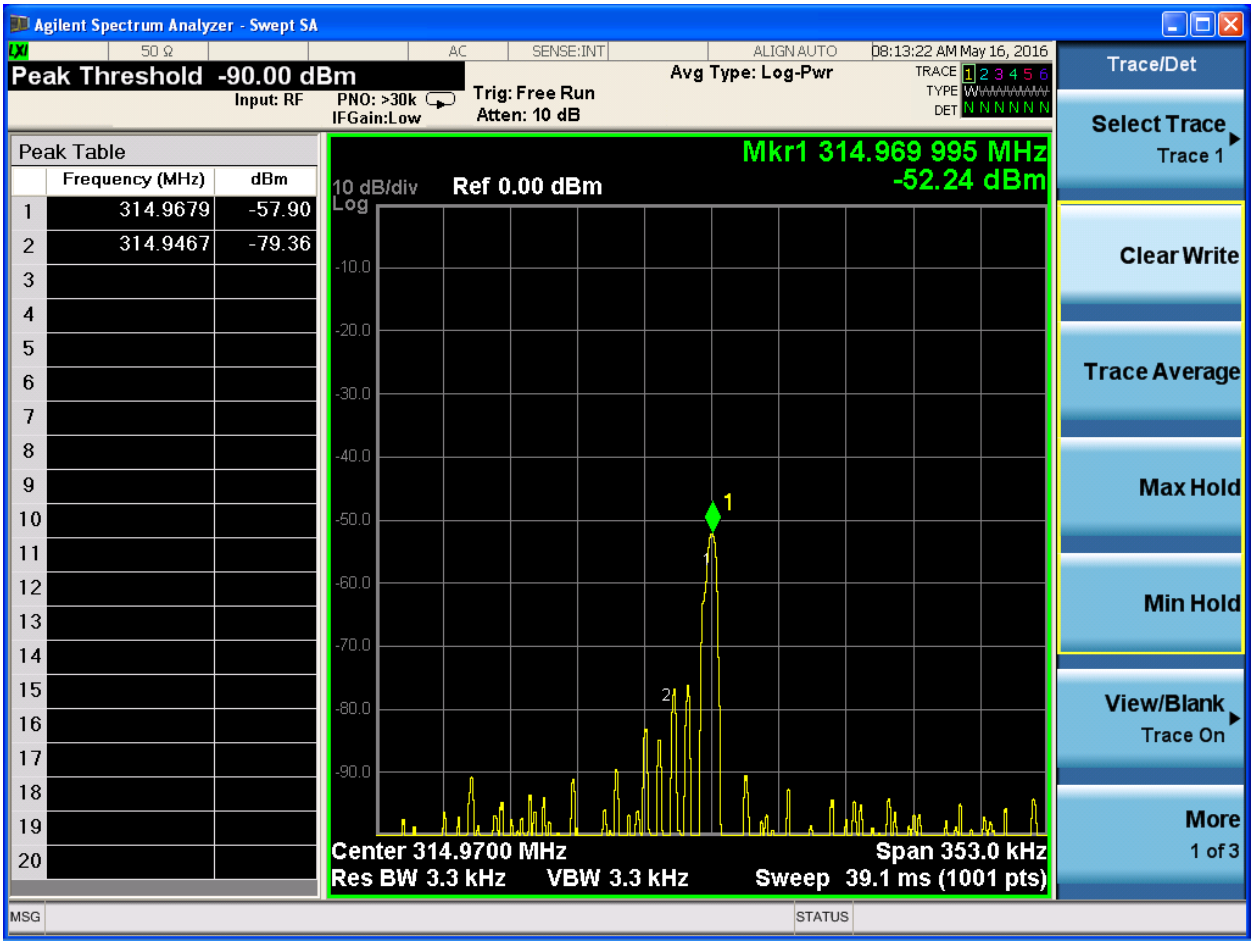


Figure 5.13: Carrier RF ASK transmitter, located approximately three feet away from the antenna connected to the input of the spectrum analyzer.

The RF transmitter specifies a maximum data rate of 8 kbps, meaning that the maximum bandwidth of the RF transmitter is 16 kHz (without considering harmonics of the square wave). The data rate used for this project was 1200 bps, which implies a bandwidth of 2400 kHz (once again ignoring other harmonics). Evidently, the bandwidth of the signal is larger than that, and more likely to be approximately 20 kHz due to the other harmonics.

The structure of the frame sent over the RF link can be seen in figure 5.13

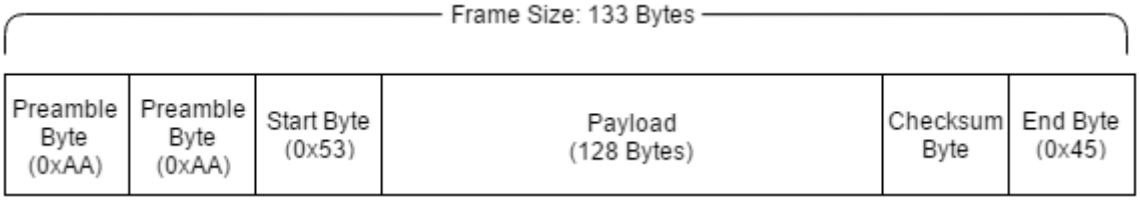


Figure 5.14: Frame format for transmission.

Despite what is indicated in figure 5.13, the payload of the frame is variable (in code). Typically, this value is 128, however dropping the payload to 64 bytes or increasing it to 256 Bytes is also feasible.

The frame begins with two pre-amble / synchronization bytes, which serve the purpose of 'ironing out' any glitches with the RF link. The RF link will usually slip up on the first one or possibly two bytes of data sent over it after there hasn't been activity on the link for a short while. Since there is a delay between frames due to the sampling of data by the microcontroller, the preamble is necessary in order to prevent the start byte from being distorted. The next byte after the preamble is the start byte. This byte indicates to the computer at the receiver end, that the frame is inbound, and that the next bytes are data bytes. The receiver is programmed to look for 128 data bytes. The checksum for the payload follows. This value is checked against the sum of the received payload in order to ensure both are equivalent. The final byte is an end flag, which serves the purpose of indicating to the user when the frame is over, and also makes it easier to find and measure the checksum value when observing the signal on the oscilloscope.

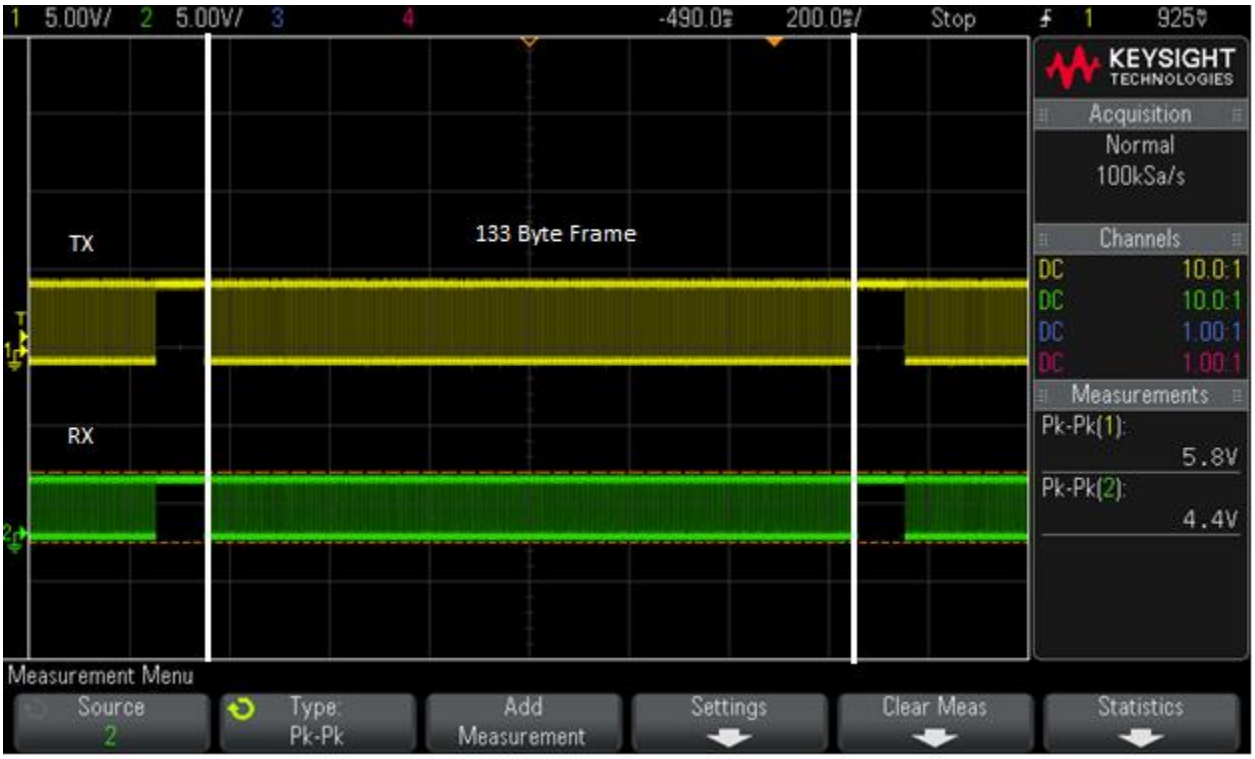


Figure 5.15: Transmitted frame (Ch1) vs received frame (Ch2) on the RF link.

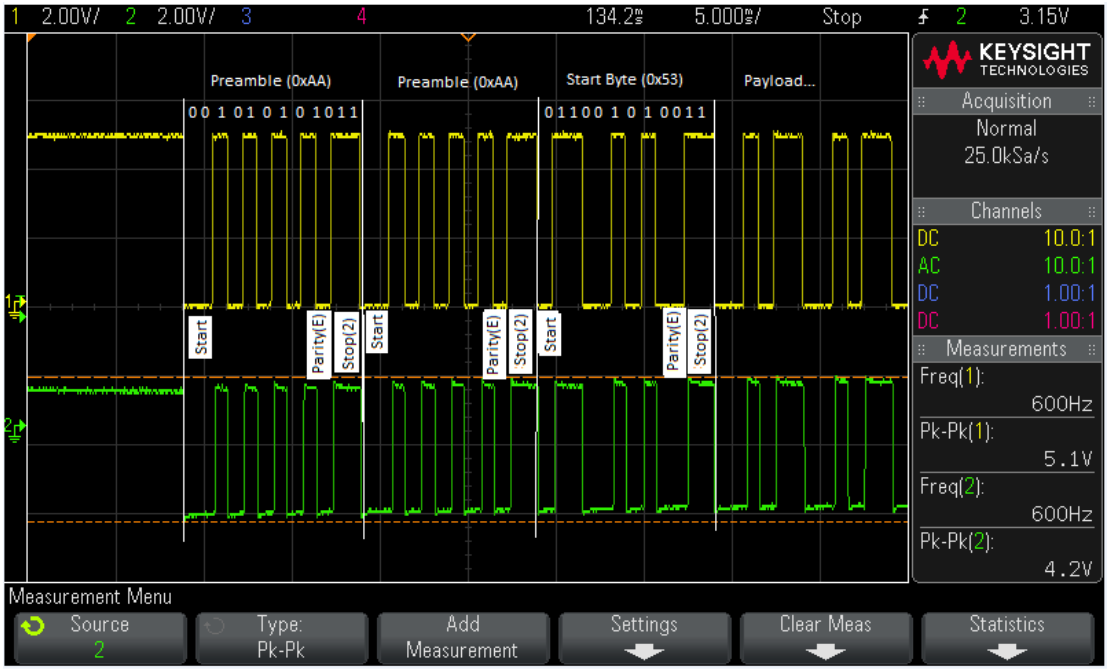


Figure 5.16: Beginning of the frame (Preamble, Preamble, Start Byte, and Payload). TX on ch1, RX on ch2.

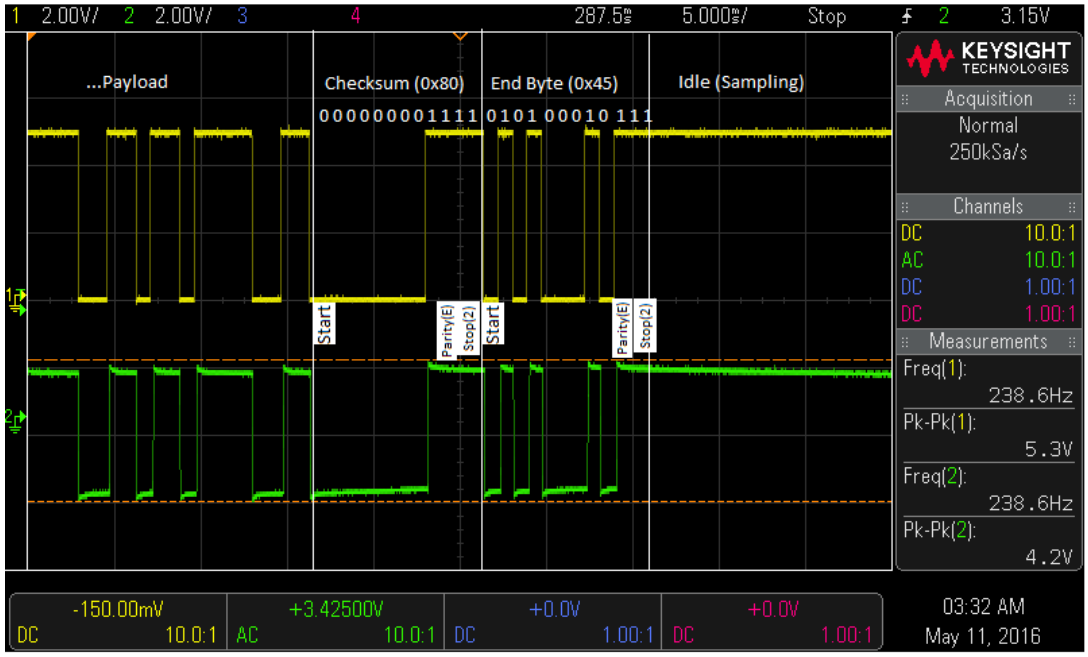


Figure 5.17: End of the frame (Payload, Checksum, and End byte). TX on ch1, RX on ch2.

The measured frame is correct according to the frame configurations done in the sampling and transmitting program.

5.6-Power Supply:

In order to supply power to the radar station and the radar module, a linear power supply producing several different voltage levels was used. In particular, the voltages produced are 12V, 5V, 3.3V and 3V. The design of the power supply can be seen in figure 5.15:

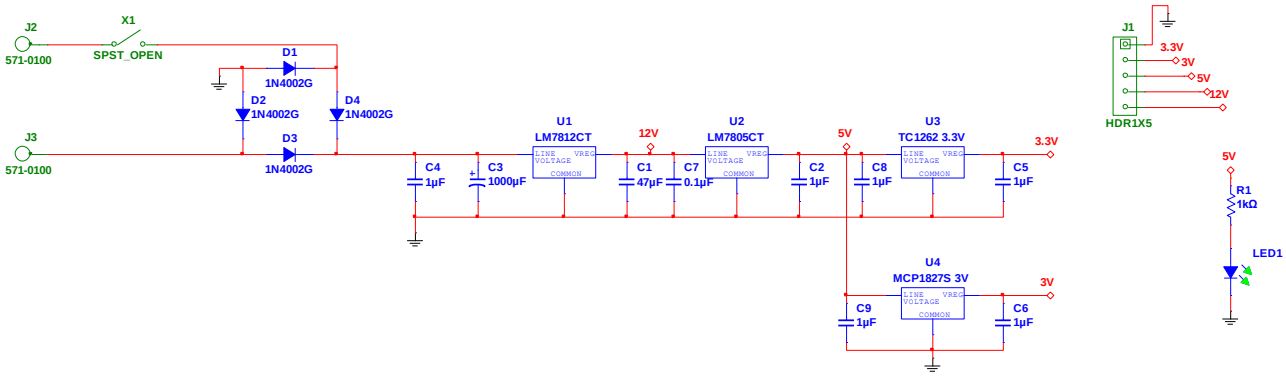


Figure 5.18: Linear power supply schematic diagram.

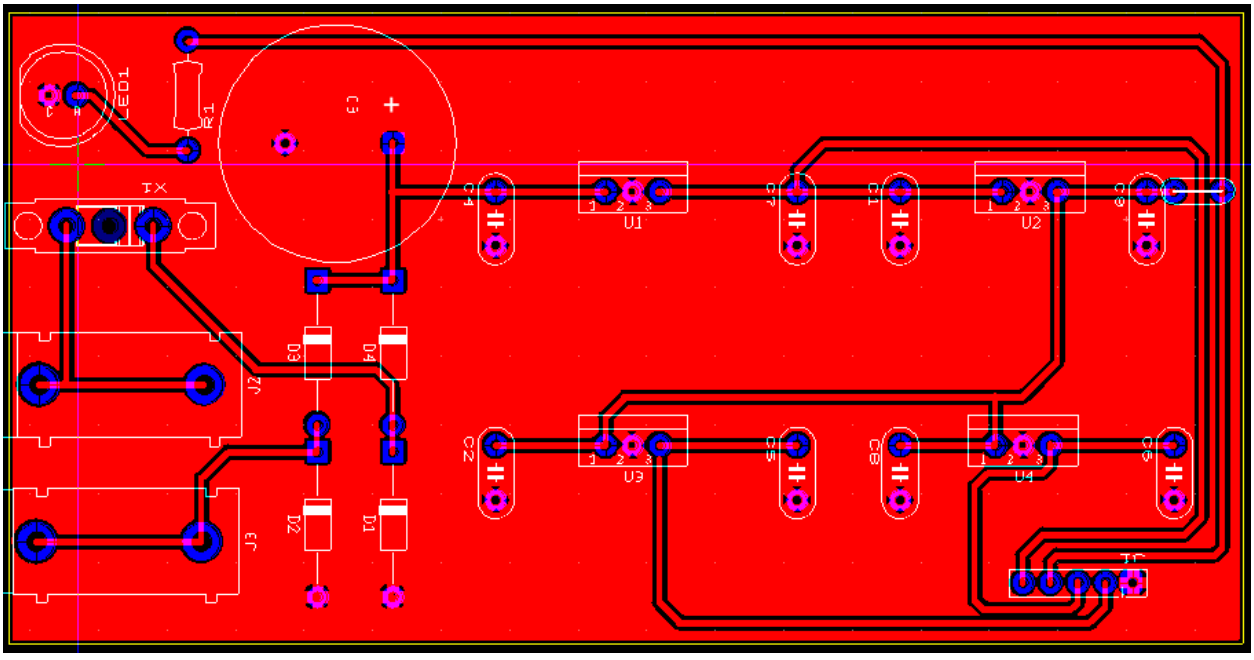


Figure 5.19: Power supply single sided PCB design.

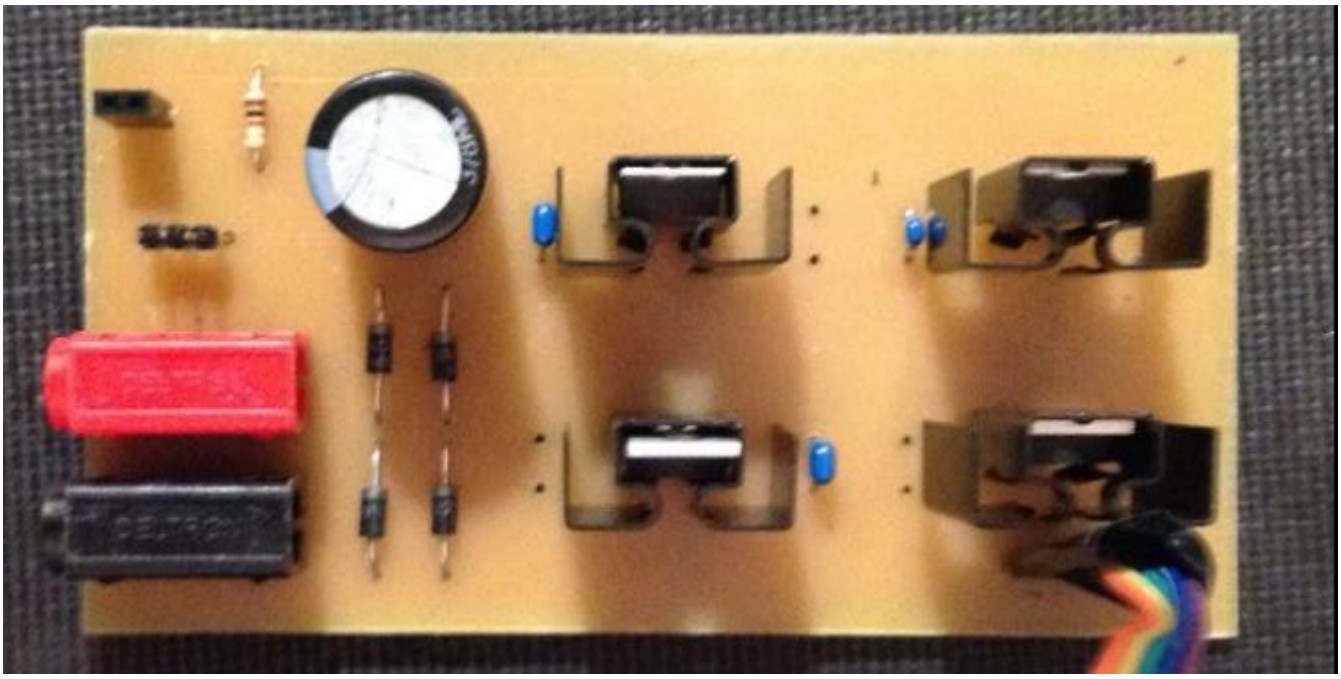


Figure 5.20: Top side of the power supply PCB.

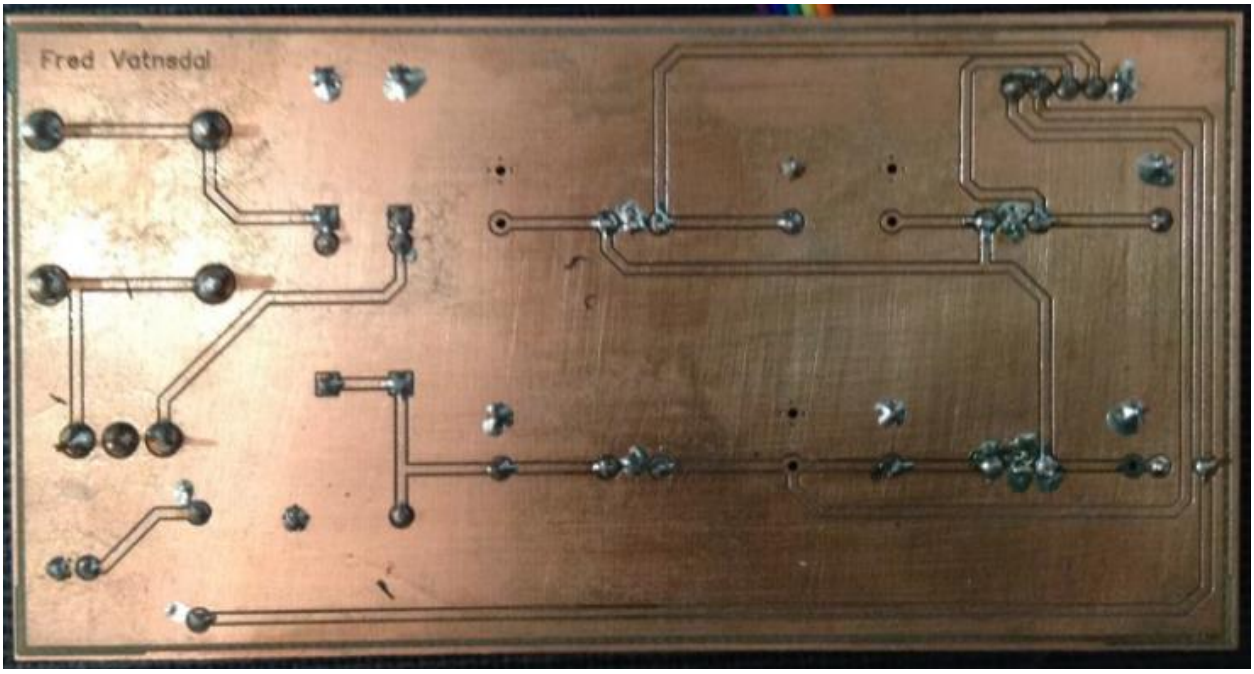


Figure 5.21: Bottom side of the power supply PCB.

Figure 5.22: Inside of the power supply unit.

A transformer with at least a 10 to 1 turn ratio steps the voltage down. The primary coil of the transformer is connected to the wall outlet, and the secondary coil is connected to the power input of the

linear power supply circuit. A 2A fuse is positioned between the primary and the wall outlet, to prevent the circuit from drawing more than 2A. The secondary connected through a three position switch, to a full bridge rectifier, which converts the AC signal into a pulsating DC signal. This pulsating DC is flattened using a large 1000uF capacitor, and then regulated to 12V. The output of the 12V regulator is stepped down to 5V, as well as being connected to a power output pin so that it may be connected to the radar station. The regulated 5V is used to create the 3.3V and 3V voltage levels for the radar module.

The efficiency of each regulator can be calculated using equation 3.6.

Equation 5.6:

$$\eta = \frac{Pout}{Pin} * 100 = \frac{Vout * Iout}{Vin * Iin} * 100$$

For the 12V regulator, under full load of the system:

$$\eta = \frac{12V * 0.22A}{15.2V * 0.22A} * 100 = 79\ \%$$

For the 5V regulator, under full load of the system:

$$\eta = \frac{5V * 0.19A}{11.9V\ * 0.19A} * 100 = 42\ \%$$

For the 3.3V regulator, under full load of the system:

$$\eta = \frac{3.3V * 0.150A}{5V * 0.150A} * 100 = 66\ \%$$

For the 3V regulator, under full load of the system:

$$\eta = \frac{3V * 0.02A}{5V * 0.02A} * 100 = 60\ \%$$

The average efficiency is as follows:

$$\eta_{avg} = \frac{79 + 42 + 66 + 60}{4} = 61.75\ \%$$

The average efficiency is 61.75 %. This is expected for a linear power supply, as typically the efficiency of these power supplies are lower than that of a switching power supply. The benefit of a linear power supply, and the reason it was chosen over a switching power supply, is due to the lower amount of noise the power supply generates.

Not all capacitors provisioned for on the PCB design were actually soldered on to the PCB. The reason for this, was that more capacitor foot prints than necessary were provided for, however they were only used if needed. C1, C3, C4, C5, C6 and C9 were the capacitors used in the final build.

Section 6.0 - Receiver Station

The purpose of the receiver station is to capture the transmitted data, using an RF receiver module, convert this data into RS232 format, and send the data over a connected serial cable, to a computer. The design was intended to be compact and portable. This maintains the theme of portability perpetuated by the wireless RF link design choice.

6.1 - Module Overview:

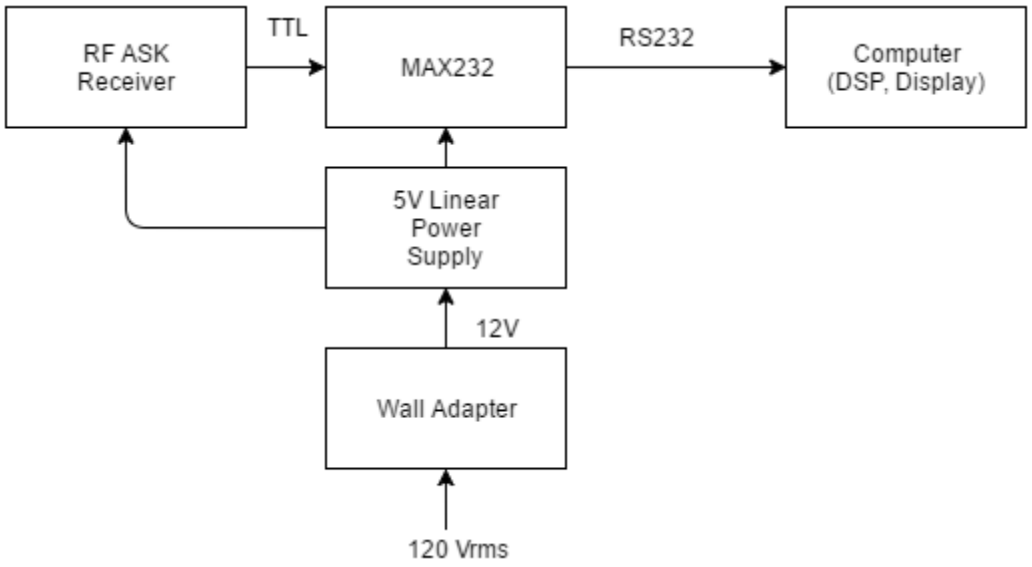


Figure 6.1: Block diagram of the receiver module.

The receiver station is quite simplistic compared to the other elements of the system as there is no processing being done on this board; however it is acting as an extension of the computer, which does all of the heavy processing on the data. In future designs, the wall adapter and linear power supply would be replaced with USB power, and the serial link between the RF receiver and the computer would be USB. This would further increase portability, and streamline the design.

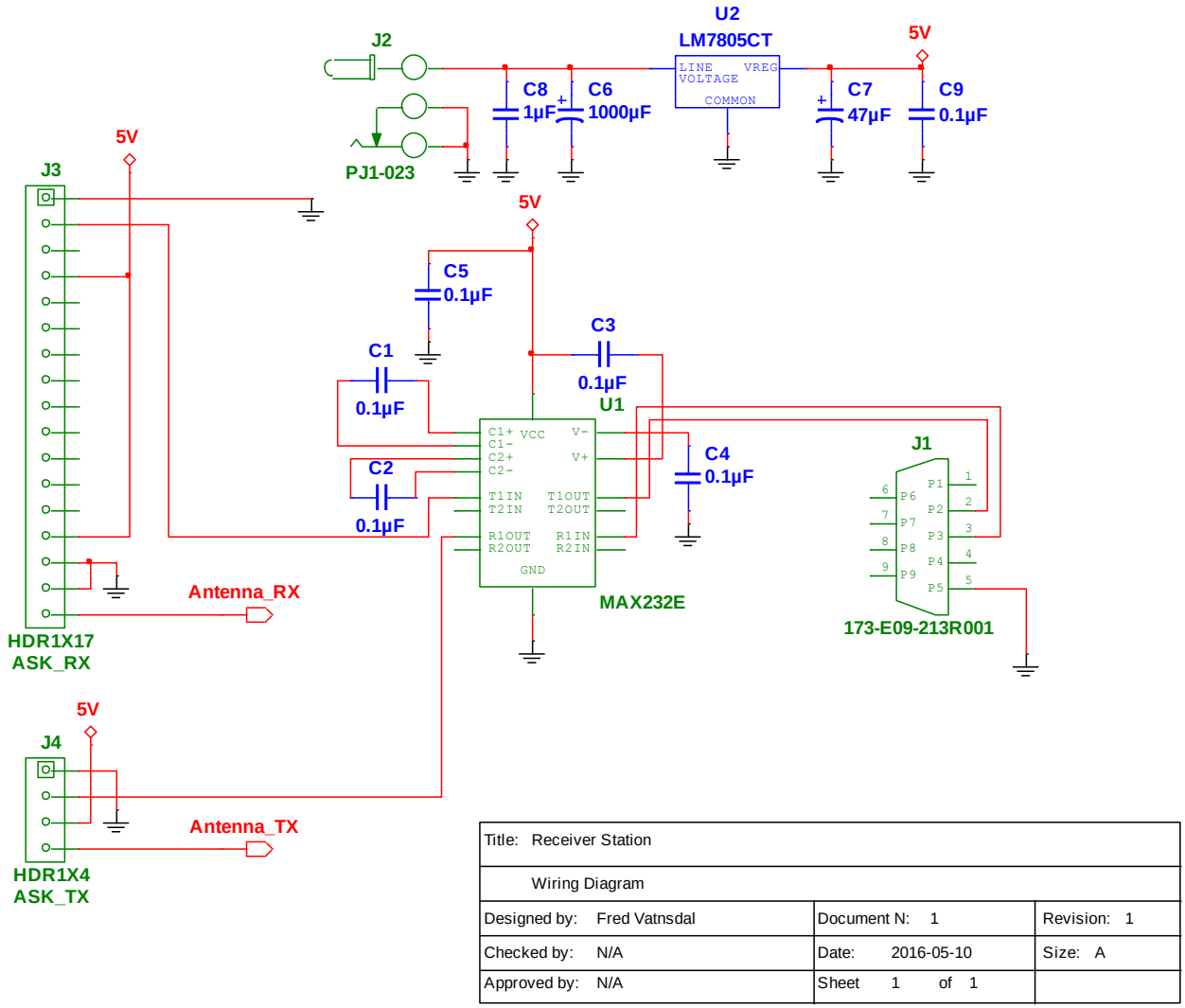


Figure 6.2: Schematic/Wiring Diagram of the receiver module.

Note: Since a large portion of this circuit is composed of connectors (some used to represent the TX and RX modules), the schematic and wiring diagram were combined.

Just like on the radar station, a full duplex communication link was provisioned for, however is unused in the final design. This would have permitted the user to type commands on the computer, and control the aspects of the radar system wirelessly. The future iterations of the design may include this added feature.

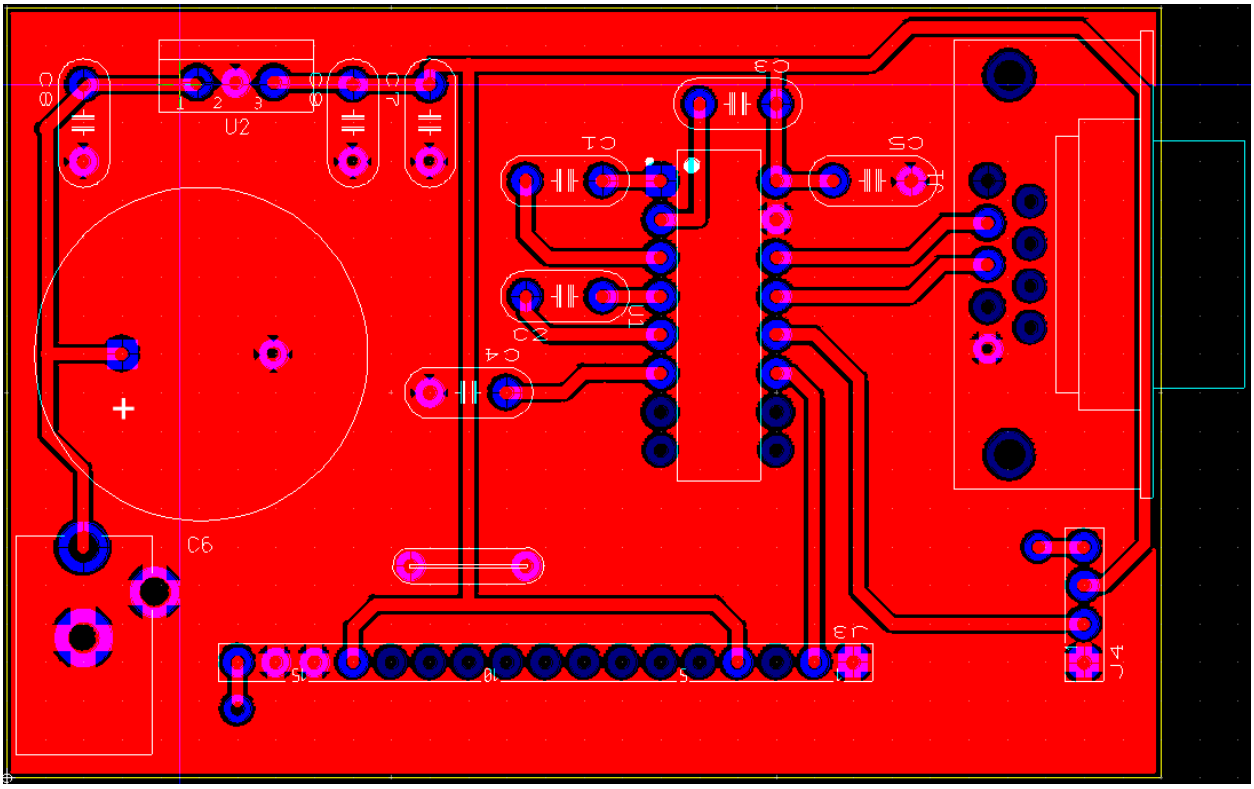


Figure 6.3: Receiver module single sided PCB design.

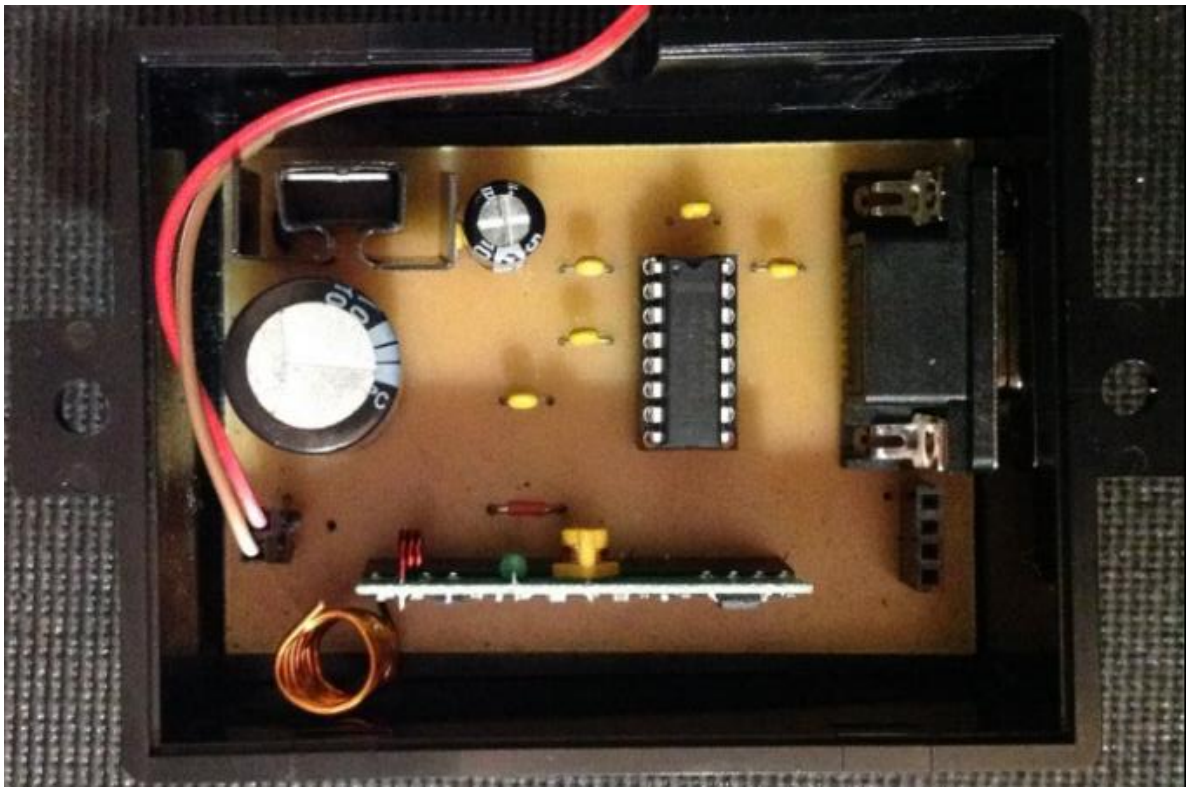


Figure 6.4: Top side of the PCB.

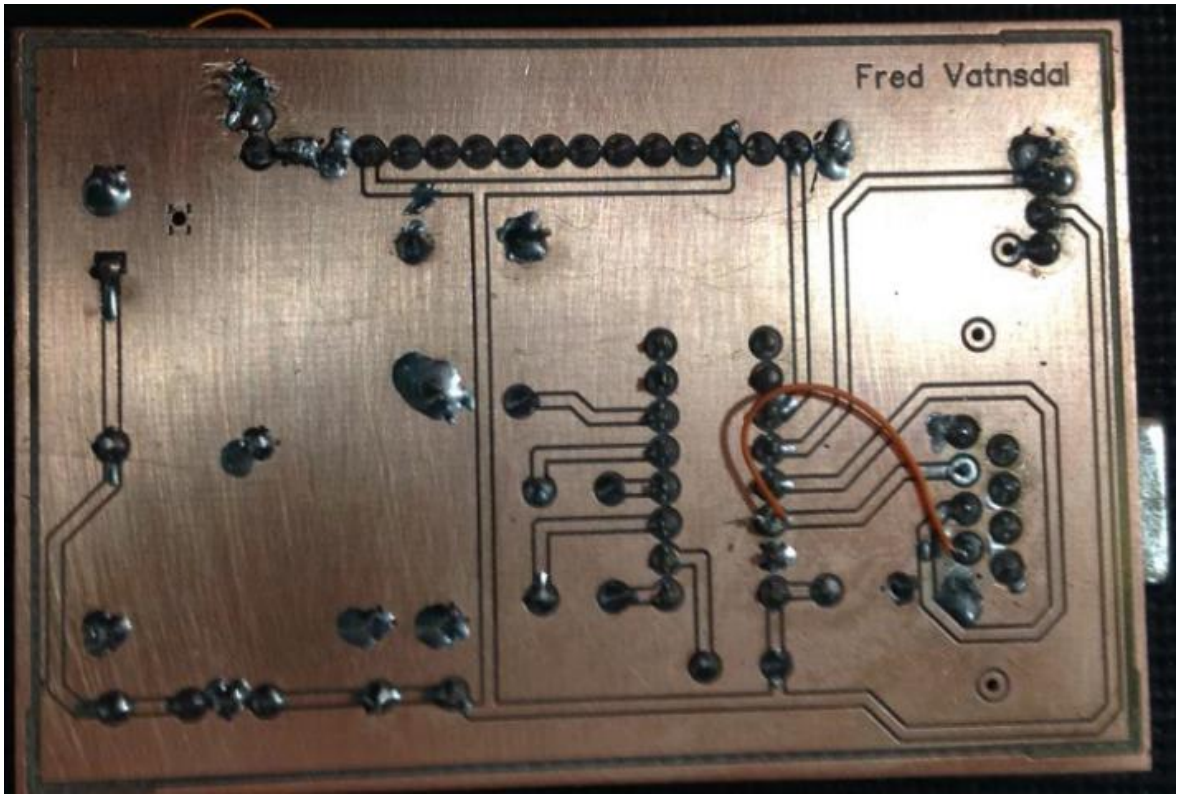


Figure 6.5: Bottom side of the receiver PCB.

This PCB design is actually slightly different than the version shown in figure 6.3. A slight mix up of the pinout of the DB9 connector resulted in needing to solve this issue with wire wrap wire.

The core component of the receiver module is RF receiver itself. This receiver was discussed briefly in Section 5.5.

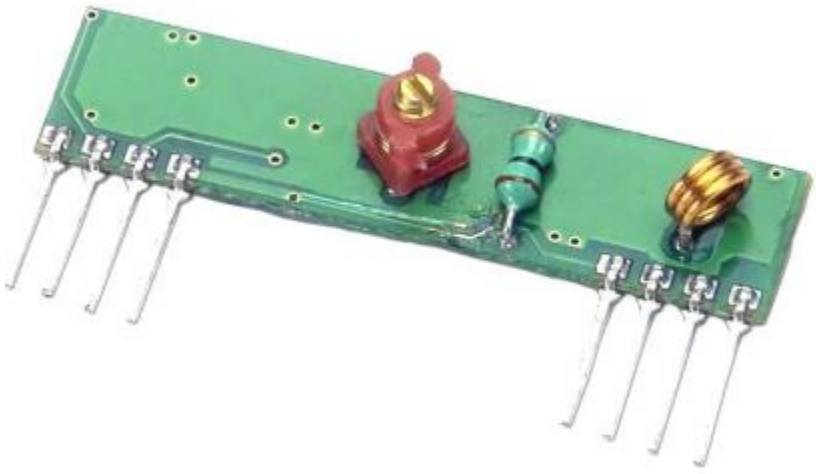


Figure 6.5: RWS-374 RF ASK receiver.

The characteristics of the receiver module are as shown in table 6.1.

Table 6.1: Electrical Characteristics of RWS-374 RF ASK receiver (from the data sheet).

Note: By 'Baseboard Data Rate', 'Baseband Data Rate' is meant.

Characteristic	Sym	Min	Type	Max	Unit
Operating Radio Frequency	FC	314.500	315.000	315.500	MHz
Sensitivity	Pref.	-106	-108	-110	dBm
Channel Width		-500		+ 500	KHz
Noise Equivalent BW	NEB		5	4	
Baseboard Data Rate				3	KB/S
Receiver Turn On Time				3	ms

The receiver actually has a lower data rate than the transmitter, accepting only 3kbps. This means that in reality, only up to 3kbps can be transmitted over the RF link. The sensitivity of the receiver module is indicated in table 6.1 (taken from the data sheet) to be typically -108dBm. This is quite good sensitivity, and it is unlikely that is the practical sensitivity.

One important note is the receiver turn on time, which is indicative of how long the receiver takes to transition from an idle state into an operational state. This is why at least one preamble byte lasting 3 of more milliseconds must be sent over the RF link before any start flags are sent after it.

Section 7.0 – Digital Signal Processing

In order to determine the distances of objects with the radar system, some form of Digital Signal Processing (DSP) is required. The objective of the DSP software, was to display a time domain and frequency domain representation of the signal, in a live updating graph, as well as the printing of the distance results for all detected objects into the console. The DSP software was written in Python 3.5, and makes use of the matplotlib, pyserial, and numpy libraries. The software, as well as the libraries it uses will be discussed in this section.

7.1 – Receiving Data:

In order to receive data, the function 'get_data' was used. A snippet from this function can be seen in code snippet 7.1. In the code snippet, the first part of the function can be seen. The first byte that is looked for is the start byte, 'S'. A try except statement is used in order convert the data into a decimal value, and catches a type error if it is thrown.

```
def get_data(ser):
    data_buff = []
    while True:
        try:
            input = ser.read(1)                 #Read the start byte
            if ord(input) == 83:                #Wait for start character ('S')
                print('--Start of frame detected--')
                break
            else:
                print('Waiting for start of frame:', ord(input))
        except TypeError:                       #Handle TypeError exception
            print('Bad Data:',input)
            continue
```

Code Snippet 7.1

The same methods are used in acquisition of the other bytes in the frame as is shown in code snippet 7.1. Getting data is done with class provided from the pyserial library. The serial object 'ser', is what is used to read bytes from the selected COM port. This object is initialized in the 'init_ser' function. The selected baud rate was 1200, with even parity, and two stop bits. The COM port is selected by the

user in the initialization stage of the software. Only valid COM port numbers are accepted to reduce the amount of exceptions that may be thrown.

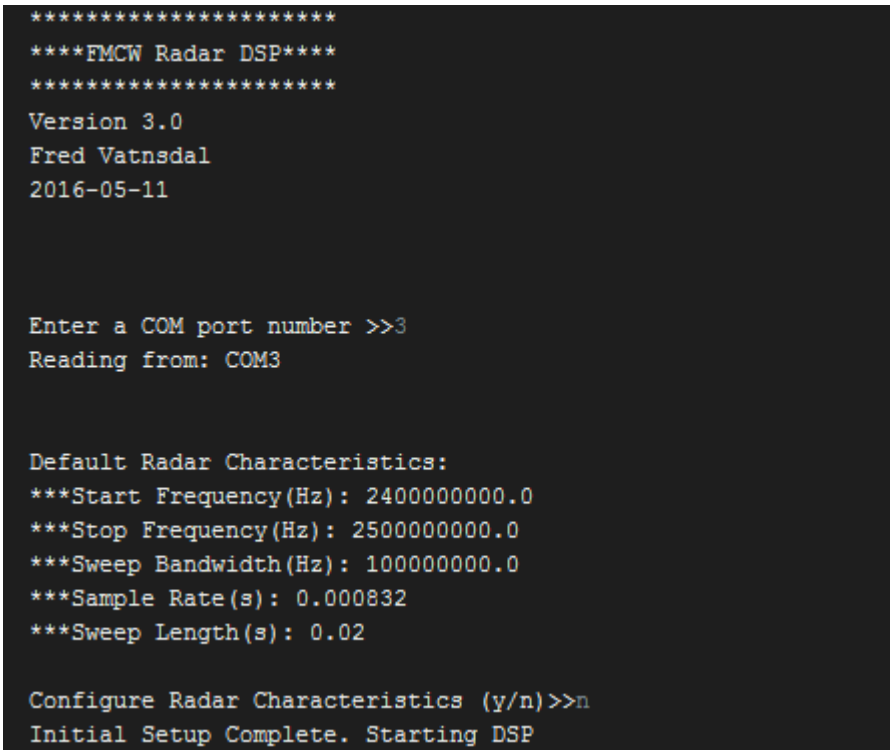


Figure 7.1: Initialization of the DSP software with default parameters.

Figure 7.1 shows the initialization of the DSP software. This involves selecting the COM port to use. In this case, the USB to serial converter was located on COM3, and as such the COM port selected was COM3.

7.2 – Computations:

In order to extract frequency information from the sampled signal, the Discrete Fourier Transform (or the improved version, the Fast Fourier Transform) can be used. The Discrete Fourier Transform is shown in equation 7.1.

Equation 7.1:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{\frac{-j2\pi nk}{N}}$$

In order to be useful, this transform must be broken down into pieces that are easier to implement in software. This can be done using Euler's equation:

Equations 7.2:

$$e^{jx} = \cos(x) + jsin(x)$$

Therefore, the equation can be broken down using the following steps:

$$X(k) = \sum_{n=0}^{N-1} x\,(n)\ \ (\cos\left(\frac{-2\pi nk}{N}\right) + jsin\left(\frac{-2\pi nk}{N}\right))$$

$$X(k) = \sum_{n=0}^{N-1} (x_{Re}(n) + jx_{im}(n)) * (\cos\left(\frac{2\pi nk}{N}\right) - jsin\left(\frac{2\pi nk}{N}\right))$$

$$X(k) = \sum_{n=0}^{N-1} (x_{Re}(n) * \cos\left(\frac{2\pi nk}{N}\right) - x_{Re}(n)jsin\left(\frac{2\pi nk}{N}\right)) + (jx_{im}(n) * \cos\left(\frac{2\pi nk}{N}\right) + (j)(-j)x_{im}(n)sin\left(\frac{2\pi nk}{N}\right))$$

$$X_{Re}(k) = \sum_{n=0}^{N-1} x_{Re}(n)\cos\left(\frac{2\pi nk}{N}\right) + \ x_{im}(n)\sin(\frac{2\pi nk}{N})$$

$$X_{im}(k) = \sum_{n=0}^{N-1} -j(x_{Re}(n) * \sin\left(\frac{2\pi nk}{N}\right) + x_{im}(n)cos\left(\frac{2\pi nk}{N}\right))$$

However, only the real portion of the signal will be sampled. Here are the two equations used in the DFT.

Equation 7.3:

$$X_{Re}(k) = \sum_{n=0}^{N-1} x_{Re}(n) \cos\left(\frac{2\pi nk}{N}\right)$$

Equation 7.4:

$$X_{im}(k) = -\sum_{n=0}^{N-1} j(x_{Re}(n) * \sin\left(\frac{2\pi nk}{N}\right)$$

Seeing as an FFT is a form of the DFT, and the FFT is being used in the DSP software, it is important to understand at the very least the inputs and outputs of the transform in order to be able to use the function provided by the numpy library.

```
def fft(self, data, sample_period, num_sam):
    fs = 1/float(sample_period)                 #Calculate the sample rate
    f_rez = fs/num_sam                          #Calculate frequency resolution
    y_data = list(map(abs, np.fft.rfft(data)))  #Convert FFT ydata to abs vals, create list.
    temp = np.arange(len(y_data))               #Arrangement of frequency bins
    freqs = []
    count = 0
    for bin in temp:                            #For each freq bin in temp list
        freqs.append(count*f_rez)               #Append bin# * frequency resolution
        count+=1                                #Increment count
    return y_data,freqs                         #Return y data list, and freq list
```

Code Snippet 7.2

The FFT function, takes an FFT on an array of data passed to. The function in which the FFT is used can be seen in Code Snippet 7.2. The FFT will output y data in different x bins. These bins are normalized frequency values, and the bin multiplied by the frequency resolution will result in the actual frequency for that bin. The frequency resolution is determined by sampling frequency, and the number of samples taken. This can be seen in equation 7.5.

Equation 7.5:

$$f_{rez} = \frac{Sample\ Frequency}{Number\ of\ Samples}$$

Thus, if the sampling frequency is 1.2 kHz, and the number of samples taken is 128, the frequency resolution can be determine using equation 7.5:

$$f_{rez} = \frac{1.2kHz}{128} = 9.4\ Hz$$

A frequency resolution of 9.4 Hz is decent for the purposes of this project, however a better resolution could be achieved with a larger payload (for instance, 256 samples).

The absolute value of the y data is taken (as it is in complex notation originally), and a list is created with that data. The output of the function 'fft' seen in the code snippet returns the discrete frequencies and their respective amplitudes in two separate by matching lists.

7.3 – Displaying Data:

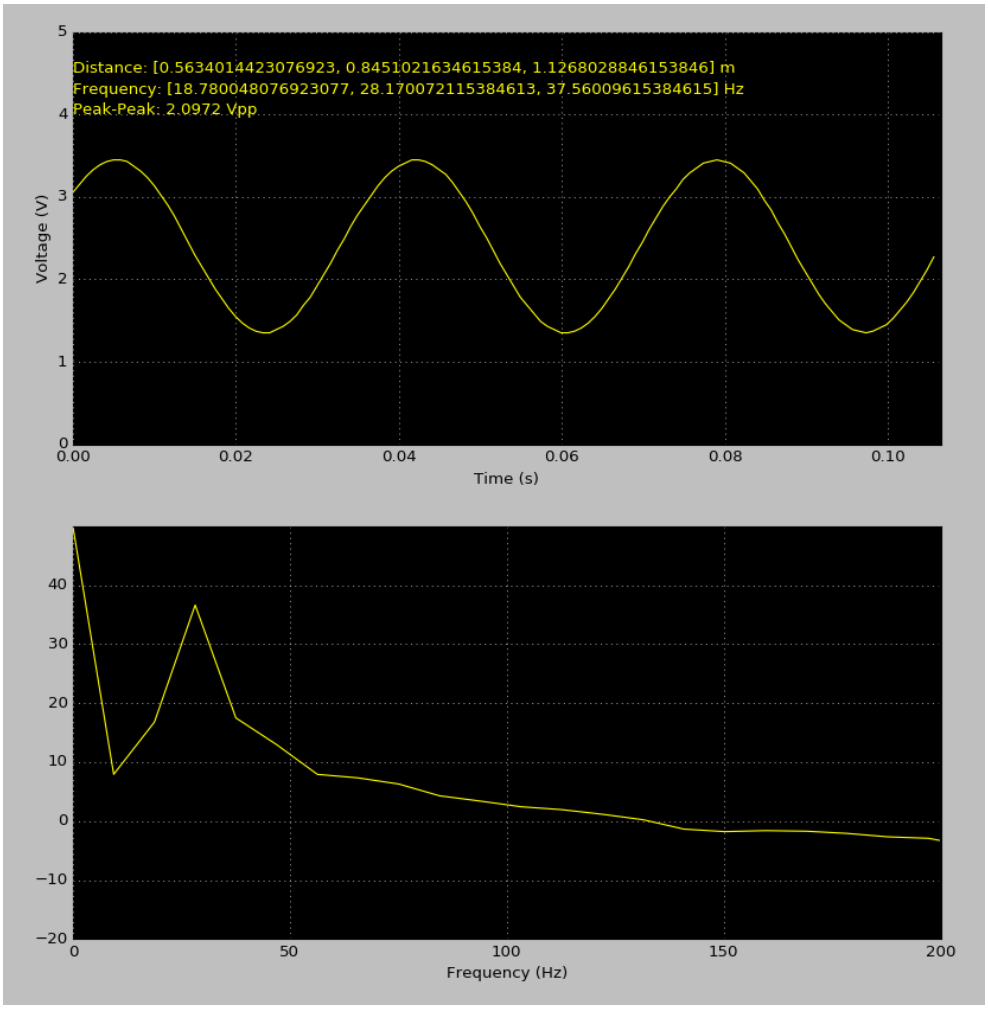


Figure 7.2: Measuring 25 Hz, and plotting using DSP software.

In order to calibrate and test the software, low frequency sinusoids were tested. A particular frequency of interest is 25 Hz, since that is typically one of the fundamental frequencies present in the output of the radar. Thus, that is the main test frequency. Figure 7.1 shows the plotting of this 25 Hz signal in the time domain and in the frequency domain. The text readout on the time domain plot is meant for the radar signal, as it displays the three most prominent frequencies detected and subsequently the distances of the objects detected. It also shows the voltage value in terms of peak to peak voltage.

The displayed frequency spectrum is quite accurate in terms of frequency, and the single harmonic present resides at nearly 25 Hz exactly (~28Hz in actuality, as that is the nearest frequency step).

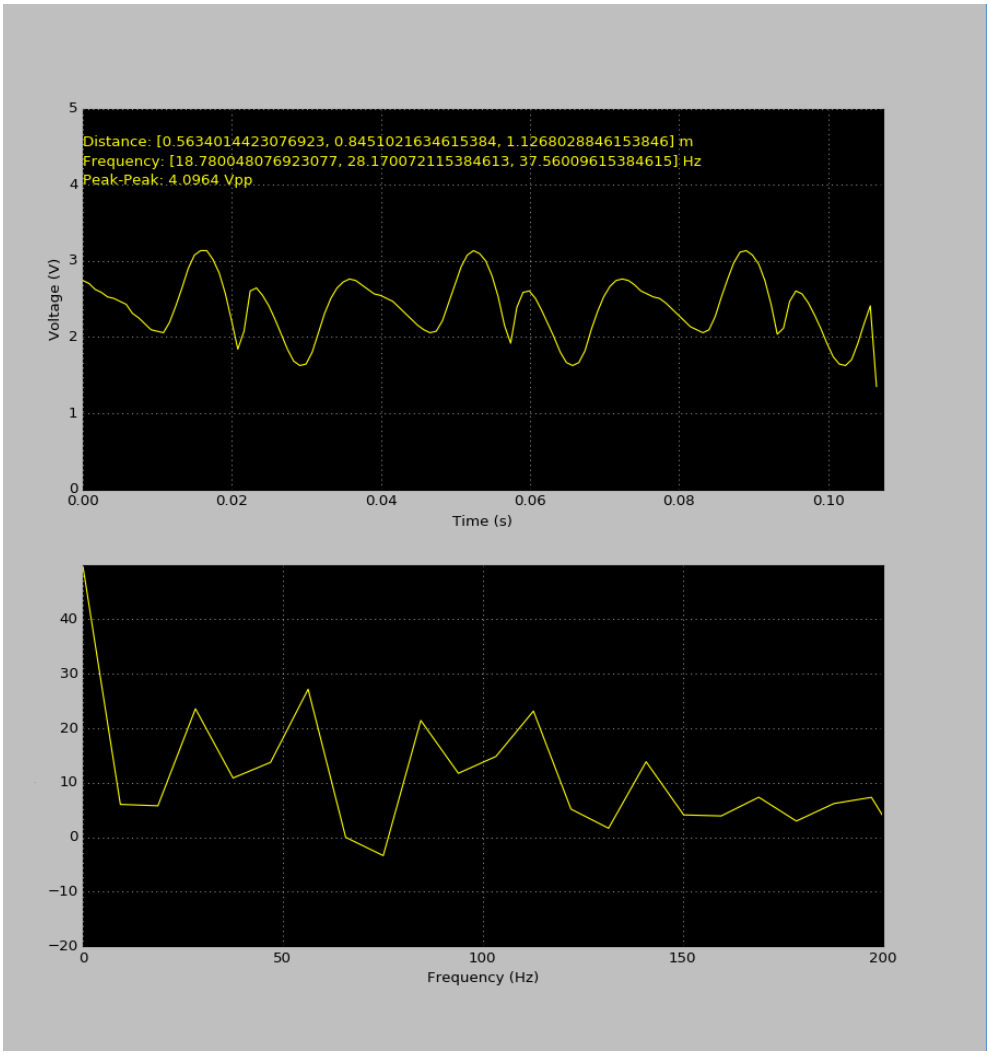


Figure 7.3: Measuring the output of the radar in both the time domain and the frequency domain.

The measured output of the radar shown in figure 7.2 contains multiple frequency components shown in the frequency spectrum. There are significant (above 15 dB) harmonics at ~25 Hz, ~50Hz, ~75Hz, and ~110 Hz. These frequencies are used to calculate the distances of detected objects. The radar scan took place in a cluttered room, and there are multiple objects being detected.

```
Significant frequencies: [56.340144230769226, 84.51021634615384, 112.68028846153845, 140.85036057692307]
Distances of Detected Objects: [1.6902043269230769, 2.5353064903846154, 3.3804086538461537, 4.225510817307692]
```

Figure 7.3: Distance display printed to console (not of the same spectrum as shown in figure 7.2).

The distance calculation is shown in equation 4.2. Detected objects are at 1.6m, 2.5m, 3.4m and 4.2m.

7.4 – Code:

FMCW DSP Software (written in Python)

```
import datetime
import serial
from matplotlib import pyplot as plt
from matplotlib import animation
from matplotlib.lines import Line2D
import numpy as np
import math
BAUDRATE = 1200
SOL = 3e8

def main():
    freq_start = 2.4e9
    freq_stop = 2.5e9
    sample_period = 832e-6
    sweep_length = 20e-3

    print('**********************')
    print('****FMCW Radar DSP****')
    print('**********************')
    print('Version 3.0')
    print('Fred Vatnsdal')
    print(datetime.date.today(),'\n','\n','\n')
    port = 0
    #Determine the COM port to be used.
    while True:
        selected_port = input('Enter a COM port number >>')
        try:
            selected_port = int(selected_port)
        except(ValueError):
            print(selected_port,'is not a number. Please enter a number.')
```

```
        continue
    if((selected_port >= 0) and (selected_port < 5)):
        break
    else:
        print('COM', selected_port, 'is out of range. Choose a number from 0 - 4.')

port = 'COM' + str(selected_port)
ser = init_ser(port)
print('Reading from:', port)

print()
print()
print('Default Radar Characteristics:')
print('***Start Frequency(Hz):', freq_start)
print('***Stop Frequency(Hz):', freq_stop)
print('***Sweep Bandwidth(Hz):', freq_stop-freq_start)
print('***Sample Rate(s):', sample_period)
print('***Sweep Length(s):', sweep_length)
print()


#Menu System for configuring sweep parameters.
while True:
    confopt = input('Configure Radar Characteristics (y/n)>>')
    if confopt == 'y':
        print()
        print('      Options Menu    ')
        print('======================')
        print('1...Start Frequency(Hz)')
        print('2...Stop Frequency(Hz)')
        print('3...Sample Rate(s)')
        print('4...Sweep Length(s)')
        print('e to exit')
```

```
while True:
    print()
    selection = input('Select a menu option (1-5)>>')
    if selection == '1':
        while True:
            temp = input ('Enter a start frequency in Hz, or e to exit >>')
            print('Selected value:',temp)
            if temp == 'e':
                break
            try:
                freq_start = float(temp)
                break
            except TypeError:
                print('Bad Input')

    elif selection == '2':
        while True:
            temp = input ('Enter a stop frequency in Hz, or e to exit >>')
            print('Selected value:',temp)
            if temp == 'e':
                break
            try:
                freq_stop = float(temp)
                break
            except TypeError:
                print('Bad Input')

    elif selection == '3':
        while True:
            temp = input ('Enter a sample rate in s, or e to exit>>')
            print('Selected value:',temp)
            if temp == 'e':
                break
```

```
            try:
                sample_period = float(temp)
                break
            except TypeError:
                print('Bad Input')

        elif selection == '4':
            while True:
                temp = input ('Enter a sweep length in s, or e to exit >>')
                print('Selected value:',temp)
                if temp == 'e':
                    break
                try:
                    sweep_length = float(temp)
                    break
                except TypeError:
                    print('Bad Input')
        elif selection == 'e':
            break;
        else:
            print('Bad input:',selection)
    break
  elif confopt == 'n':
    break
  else:
    print('Bad input:',confopt)

print('Initial Setup Complete. Starting DSP')
ani = Plot(ser, sample_period, freq_start, freq_stop, sweep_length)    #Create the plot object
plt.show()                                          #Show the plot
print('Plot Created')
```

```
def init_ser(port):
    try:
      return serial.Serial(port=port,
                    baudrate=1200,
                    bytesize=serial.EIGHTBITS,
                    parity=serial.PARITY_EVEN,
                    stopbits=serial.STOPBITS_TWO,
                    timeout=0.1)

    except(serial.serialutil.SerialException):
        print('Error: The serial port', port, 'is in use, or is invalid') #Error message
        print('Exiting...')
        exit(0)



def get_data(ser):
   data_buff = []
   while True:
      try:
        input = ser.read(1)              #Read the start byte
        if ord(input) == 83:             #Wait for start character ('S')
          print('--Start of frame detected--')
          break
        else:
          print('Waiting for start of frame:', ord(input))
      except TypeError:                  #Handle TypeError exception
        print('Bad Data:',input)
        continue
   num_sam = 128
   points = num_sam
   num_sam = points
   cnt = 0
   checksum = 0
```

```
while points > 0:
    input = ser.read(1)             #Read data
    try:
        value = ord(input)          #Try conversion to base 10
    except TypeError:
        print('Bad Data:',input)      #Handle exception
        continue

    data_buff.append(value*0.0196)     #Convert to voltage and append
    checksum += value
    points -= 1
    cnt += 1
input = ser.read(1)
try:
    verify_chksum = ord(input)
except TypeError:
    print('Invalid Checksum:',verify_chksum)
checksum = hex(checksum)
verify_chksum = hex(verify_chksum)
if (checksum and verify_chksum) == verify_chksum:
    print('Checksum OK')
    print('Checksum:',(checksum and verify_chksum), 'Verify Checksum:',verify_chksum)
else:
    print('Error - Bad Checksum')
    print('Checksum:',checksum, 'Verify Checksum:',verify_chksum)
input = ser.read(1)
try:
    end = ord(input)
    if end == 69:
        print('--End  of frame detected--')
    else:
        print('Expected end of frame, got:',end)
except TypeError:
```

```
        print('Bad Data:',input)

    return data_buff, num_sam            #Return the data buffer and the number of samples received


class Plot(animation.TimedAnimation):
    def __init__(self, ser, sample_period, freq_start, freq_stop, sweep_length):
        self.ser = ser                          #Serial interface
        self.sample_period = sample_period
        self.freq_start = freq_start
        self.freq_stop = freq_stop
        self.sweep_length = sweep_length
        self.constant = ((self.freq_stop - self.freq_start)/sweep_length)
        self.dist_val = 0                       #Distance text value
        self.freq_val = 0                       #Frequency text value
        self.vpp_val = 0                        #Amplitude text value
        self.data_y, self.num_sam = get_data(ser)

        #Init the figure and time plot.
        fig = plt.figure()
        self.axis_time = fig.add_subplot(2, 1, 1)    #Add the subplot (whole plot area)
        self.axis_freq = fig.add_subplot(2, 1, 2)
        self.axis_time.set_axis_bgcolor('black')     #Set the background of the plot to black
        self.axis_freq.set_axis_bgcolor('black')
        self.axis_time.grid()                   #Enable plot grid
        self.axis_freq.grid()
        ax = self.axis_time.get_xgridlines()        #Acquire x grid lines for color change
        ay = self.axis_time.get_ygridlines()        #Acquire y grid lines for color change
        for l in ax:                            #Set grid color to white
            l.set_color('white')
        for l in ay:
            l.set_color('white')
        ax = self.axis_freq.get_xgridlines()        #Acquire x grid lines for color change
```

```
    ay = self.axis_freq.get_ygridlines()        #Acquire y grid lines for color change
    for l in ax:                                #Set grid color to white
        l.set_color('white')
    for l in ay:
        l.set_color('white')
    self.count = 0

    self.d_text = self.axis_time.text(0, 4.5, ' Distance (m): 0', color='yellow',fontsize=12)     #Initialize distance text
    self.f_text = self.axis_time.text(0, 4.25, ' Frequency (Hz): 0', color='yellow',fontsize=12)  #Initialize frequency text
    self.a_text = self.axis_time.text(0, 4, ' Peak-Peak (Vpp): 0', color='yellow',fontsize=12)    #Initialize amplitude text

    self.t = np.linspace(0, 80, 400)                                #Setup time
    self.x = np.arange(len(self.data_y)) * sample_period                    #Sample
    self.y = self.data_y                                        #Amplitude data (time domain)
    self.xf = self.fft( self.data_y ,self.sample_period, self.num_sam)       #Compute FFT
    self.axis_time.set_xlabel('Time (s)')
    self.axis_time.set_ylabel('Voltage (V)')
    self.line1 = Line2D([], [], color='yellow')
    self.axis_time.add_line(self.line1)                             #Add the line to the data
    self.axis_time.set_xlim(0, len(self.data_y) * sample_period)            #Set x-lim (0 - final t)
    self.axis_time.set_ylim(0, 5)                                   #Set y-lim (0V to 5V)

    self.axis_freq.set_xlabel('Frequency (Hz)')
    self.axis_freq.set_ylabel('Log Scale')
    self.line2 = Line2D([], [], color='yellow')
    self.axis_freq.add_line(self.line2)
    self.axis_freq.set_xlim(0, 200)
    self.axis_freq.set_ylim(0, 100)

    animation.TimedAnimation.__init__(self, fig, interval=1, blit=True)      #Setup timed animation

def _draw_frame(self, framedata):
    i=framedata
```

```
    self.line1.set_data(self.x[:i], self.y[:i])      #Update line1 data from x,y values
    self.line2.set_data(self.xf[:i], self.yf[:i])

    self._drawn_artists = [self.line1, self.line2]   #Draw lines

def new_frame_seq(self):
    return iter(range(self.t.size))

def _init_draw(self):
    self.line1.set_data([], [])                  #Clear line data
    self.line2.set_data([], [])
    self.update_data()                           #Update data
    self.update_text()                           #Update text

def fft(self, data, sample_period, num_sam):
    fs = 1/float(sample_period)                  #Calculate the sample rate
    f_rez = fs/num_sam                           #Calculate frequency resolution
    y_data = list(map(abs, np.fft.rfft(data)))       #Convert FFT ydata to abs vals, create list.
    temp = np.arange(len(y_data))                #Arrangement of frequency bins
    freqs = []
    count = 0
    for bin in temp:                             #For each freq bin in temp list
        freqs.append(count*f_rez)                #Append bin# * frequency resolution
        count+=1                                 #Increment count
    return y_data,freqs                          #Return y data list, and freq list

def distance_calculation(self, xf, yf):
    sig_freq = []                                #Signifcant frequencies (harmonics above a certain amplitude)
    d = []                                       #Distance values

    for freq in xf:
        if yf[xf.index(freq)] > 15 and freq > 0:
```

```
            sig_freq.append(freq)                #Add significant frequencies to list

    for freq in sig_freq:                        #Calculate distance for significant harmonics
        d.append(SOL*freq/(2*self.constant))
    return d, sig_freq                           #Return distance values

def update_data(self):
    self.data_y, self.num_sam = get_data(self.ser)

    #update time domain data
    self.x = np.arange(len(self.data_y)) * self.sample_period
    self.y = self.data_y

def update_text(self):
    #update frequency domain data
    self.yf, self.xf = self.fft(self.data_y,self.sample_period, self.num_sam)
    self.yf = [20*math.log10(y_val) if y_val > 0 else -20 for y_val in self.yf] #Convert to logarithmic scale.
    self.axis_freq.set_ylim(-20,max(self.yf))

    #update distance data
    self.d, sf = self.distance_calculation(self.xf, self.yf)

    #update graph text
    self.d_text.set_text('Distance: {0} m'.format(self.d[:3]))
    self.f_text.set_text('Frequency: {0} Hz'.format(sf[:3]))
    self.vpp_val = max(self.data_y) - min(self.data_y)
    self.a_text.set_text('Peak-Peak: {0} Vpp'.format(self.vpp_val))

    #print updates to console
    print('Spectrum Frequencies:', self.xf)
    print('xf size:', len(self.xf))
    print('Spectrum Amplitudes:',self.yf)
    print('yf size:', len(self.yf))
```

```
        print('Significant frequencies:',sf)
        print('Distances of Detected Objects:', self.d)

        return self.d_text, self.f_text, self.f_text, self.a_text,

if __name__ == '__main__': #run main() on startup
    main()
```

Section 8.0 – Conclusion

In conclusion, the majority of the objectives of this project were completed. Many new areas were explored such as RF PCB design, sampling of analog signals, and writing DSP algorithms. The FMCW radar module built was functional, and went beyond what was originally proposed. All hardware systems functioned as intended. Many different methods of digital signal processing were attempted. Originally, the DSP was going to be done on the microcontroller, however the processing was slow and slightly inaccurate. Next, MATLAB DSP was attempted using audacity to sample the analog output. This method was also discarded. Finally, the DSP program was written in Python, and the data was obtained through the com port. This proved to be the absolute best method of doing the digital signal processing between doing it in C on the microcontroller, and doing it in MATLAB. The digital signal processing is still an area that needs work. This would be one of the main areas of concentration for future improvements. In addition, the efficiency of the radar module could be improved through the use

Section 9.0 – Troubleshooting

Troubleshooting procedure:

1. Turn off the entire system, and unplug the power supply from radar station. Turn on the power supply and verify that the voltages on the respective pin are 12V, 5V, 3V, and 3.3V. If the voltage levels on one or many of the pins are not correct, the problem is with the power supply:
    - Verify connections within the power supply unit.
    - Determine if one of the linear regulators is problematic
    - Replace linear regulator if required
2. Turn off the power supply and reconnect to the system. Remove the radar module to measure the triangular waveform input to the radar.
    - Turn on the power supply, and use the oscilloscope to measure the waveform on test point 1.
    - A triangular wave with a DC offset of approximately 5V and an amplitude of 3Vpp. Adjust the trim pot R3 on the radar station until this is achieved.
    - If the triangular wave is satisfactory, turn off the system and reconnect the radar module. Turn the power supply back on.
3. Using a second channel, measure the waveform at test point 4. Turn the R24 to the 0Ω position and ensure that U5 is not hot to the touch.
    - The waveform should be a low amplitude ripple.
    - If a similar waveform is not present, check the connections of the coaxial cable connecting the radar to the antennas.
4. While still monitoring the test point 4, vary R24 until the amplitude of the waveform is approximately 1Vpp.
5. Verify that there is data being transmit by measuring the waveform at test point 6. It should appear as a TTL level serial data stream.
6. Using another probe measure the received data stream at the receiver module. This measurement is taken on pin 11 of the U1 on the receiver board.
    - An almost identical data stream to the one seen on the radar station should be seen at this pin.
    - If this is not the case, try moving the receiver module closer to the transmitter and attempt again.
    - Ensure the receiver has power.
    - Try adjusting the antenna.

7. On the computer running the python software, if ‘Bad Data’ is being transmit to console:
    - Stop the software.
    - Unplug the USB to serial cable from the computer and from the receiver module.
    - Connect the USB connector to the computer.
    - Run the software
    - Connect the serial connector to the DB9 connector on the receiver module.
    - Verify that data is being received, and that the graphs are open and updating.

Section 10.0 – Bill of Materials

Table 10.1: Bill of Materials

Component	Quantity (CAD)	Sub Cost (CAD)	Cost (CAD)
HMC385LP4	1	20.95	20.95
PAT1220	2	0.57	1.14
SST12LP14C	1	0.94	0.94
SKY16406-381LF	1	5.75	5.75
HMC667LP2	1	18.26	18.26
HMC272AMS8	1	5.75	5.75
CONSMA003	2	5.03	10.06
4 Pin Header	4	0.78	3.12
2 Pin Header	2	0.84	1.68
7 Pin Header	2	1.88	3.76
5 Pin Header	2	0.84	1.68
6 Pin Header	1	0.43	0.43
Banana Connector	2	2.49	4.98
DB9 Connector	1	2.58	2.58
40 Pin Socket	1	2.87	2.87
16 Pin Socket	3	1.23	3.69
8 Pin Socket	1	0.68	0.68
TL074	1	0.75	0.75
XR2206	1	11.40	11.40
MAX232	1	5.12	5.12
Atmega16	1	7.61	7.61
AT24C64B	1	1.20	1.20
TWS-BS	1	5.39	5.39
RWS-374	1	6.76	6.76
LM7812	1	0.87	0.87
LM7805	2	0.85	1.70
MCP1827S-3002E	1	1.65	1.65
LM3940IT-3.3	1	2.52	2.52
Capacitor (1206 SMD)	2	0.64	1.28
Capacitor (0603 SMD)	6	0.30	1.80
Capacitor (0402 SMD)	9	0.30	2.70
Capacitor polarized	6	1.04	6.24
Capacitor unpolarized	25	0.29	7.25
Resistor 1/4W	31	0.15	4.65
Resistor (0602 SMD)	1	0.30	0.30
Resistor (POT)	3	1.13	3.39

LED	2	0.78	1.56
USB to Serial	1	10.00	10.00
Transformer	1	40.00	40.00
Power Adapter	1	2.56	2.56
Jumper Wires	34	0.15	5.10
2x1'' FR408 PCB	3	20.00	60.00
Double Sided Copper	1	3.00	3.00
Single Sided Copper	1	4.50	4.50
SRW072-CB-ND (Box)	1	18.16	18.16
377-1219-ND (Box)	1	15.97	15.97
CU-1950 (Box)	1	2.18	2.18
CU-1955 (Box)	1	3.63	3.63
SMA Cable	2	15.92	31.84
SMA Termination	2	2.24	4.48
Heat Sink	5	0.59	2.95

Total (CAD):	366.83

Section 11.0 – References

Robert O'Donnell. *RES.LL-001 Introduction to Radar Systems, Spring 2007*. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu(Accessed 11 May, 2016). License: Creative Commons BY-NC-SA

"Radar Basics." *Radar Basics*. N.p., n.d. Web. 11 May 2016

Gregory Charvat, Jonathan Williams, Alan Fenn, Steve Kogon, and Jeffrey Herd.*RES.LL-003 Build a Small Radar System Capable of Sensing Range, Doppler, and Synthetic Aperture Radar Imaging, January IAP 2011*. (Massachusetts Institute of Technology: MIT OpenCourseWare), http://ocw.mit.edu (Accessed 11 May, 2016). License: Creative Commons BY-NC-SA

"6 GHz Frequency Modulated Radar." *Henriks Blog ATOM*. N.p., 02 Dec. 2014. Web. 11 May 2016.

Data Sheets

http://www.analog.com/media/en/technical-documentation/data-sheets/hmc385.pdf

http://ww1.microchip.com/downloads/en/DeviceDoc/75034A.pdf

http://www.skyworksinc.com/uploads/documents/SKY16406_381LF_202503A.pdf

http://www.analog.com/media/en/technical-documentation/data-sheets/hmc667.pdf

http://www.analog.com/media/en/technical-documentation/data-sheets/hmc272a.pdf

http://www.ti.com/lit/ds/symlink/tl074.pdf

https://www.sparkfun.com/datasheets/Kits/XR2206_104_020808.pdf

http://www.atmel.com/images/doc2466.pdf

http://cdn.sparkfun.com/datasheets/Wireless/General/RWS-374-3_315MHz_ASK_RF_Receiver_Module_Data_Sheet.pdf