Software Defined Radio Short Range Radar

Nicholas Everett Kohls

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

David G. Long, Chair
Cammy K. Peterson
Brian D. Jeffs

Department of Electrical and Computer Engineering

Brigham Young University

ABSTRACT

Software Defined Radio Short Range Radar

Nicholas Everett Kohls
Department of Electrical and Computer Engineering, BYU
Master of Science

High cost is a current problem with modern radar systems. Software-defined radios (SDRs) offer a possible solution for low-cost customizable radar systems. An SDR is a radio communication system where, instead of the traditional radio components implemented in hardware, many of the components are implemented in software on a computer or embedded system. Although SDRs were originally designed for wireless communication systems, the firmware of an SDR can be configured into a radar system. With new companies entering the market, various types of low-cost SDRs have emerged. This thesis explores the use of a LimeSDR-Mini in a short-range radar through open software tools and custom code.

The LimeSDR-Mini is successfully shown to detect targets at a short range. However, due to the instability of the LimeSDR-Mini, the consistent detection of a target is not possible. This thesis shows how the LimeSDR is characterized and how timing synchronization and instability issues are mitigated. The LimeSDR-Mini falls short of operating reliable in a radar system and other SDR boards need to be explored as viable options.

Test setups using coaxial cables and test setups using antennas in an outdoor environment show the instability of the LimeSDR-Mini. The transmitter and the receiver are asynchronous. The timing difference varies slightly from run to run, which results in issues that are exacerbated in a short-range radar. The bleed-through signal is the signal leakage from the transmitter to the receiver. The bleed-through signal prevents the detection of targets at a short-range. Feed-through nulling is a signal processing technique used to eliminate the bleed-through signal so that short-range targets can be detected. The instability of the LimeSDR-Mini reduces the effectiveness of feed-through nulling techniques.

Keywords: software defined radio (SDR), radar, linear frequency modulated continuous wave (LFMCW), ground penetrating radar (GPR), LimeSDR Mini

ACKNOWLEDGMENTS

I am grateful for all the support I have received. First and foremost, my wife, Tessa. She is the reason I will always cherish my time I spent pursuing my graduate degree. Additionally, I am also grateful for my parents; John and Carla. Their example and confidence in me perpetuated my education. I am also grateful for the hospitality of Marian and George. Lastly, I am grateful for Dr. Long. His faith in me often exceeded my own.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION AND BACKGROUND

1.1 Introduction

Radar is an electronic system that uses radio waves to detect and find the range, size, angle, and velocity of a target. Radio waves are electromagnetic radiation that have frequencies as low as 30 Hz to as high as 300 GHz. A radar transmits a modulated signal and the echoed signal from the target is received by the radar system. Signal processing techniques are used to infer information about the target. The system requirements for a radar depend heavily on the application of the radar. For example, a radar to detect a car stopped at an intersection is different than an airport radar to detect multiple airplanes.

Developing a custom radar system can be costly in both hardware and designer time. Traditionally, Radio Frequency (RF) systems are implemented using discrete hardware components (e.g., amplifiers, filters, mixers, modulators, demodulators, etc.). In accordance with Moore's law, faster digital processing hardware now exists. Instead of signal processing being implemented only in hardware, the RF systems and signal processing can now be implemented through software.

Software-defined radios (SDRs) are radio communications systems where radio components are implemented by means of software on a computer or embedded system. The behavior of SDRs can be dynamically redefined by software or firmware changes. As a result, an SDR can easily switch from using radio waves for Bluetooth communication to using radio waves for IEEE 802.11 Wi-Fi. The research in this paper explains how an SDR can be implemented as a low-cost radar.

The SDR used in this research is the LimeSDR-Mini (see Figure 1.1). It is an inexpensive (US$175) hardware platform for developing and prototyping digital and RF designs using Altera's Cyclone IV FPGA and Lime Microsystems transceiver. It has a single transmitting channel and a single receiving channel. In keeping with the small form factor of the LimeSDR-Mini, the single-board computer Raspberry Pi 4 (see Figure 1.2) is used to interface with the SDR. The

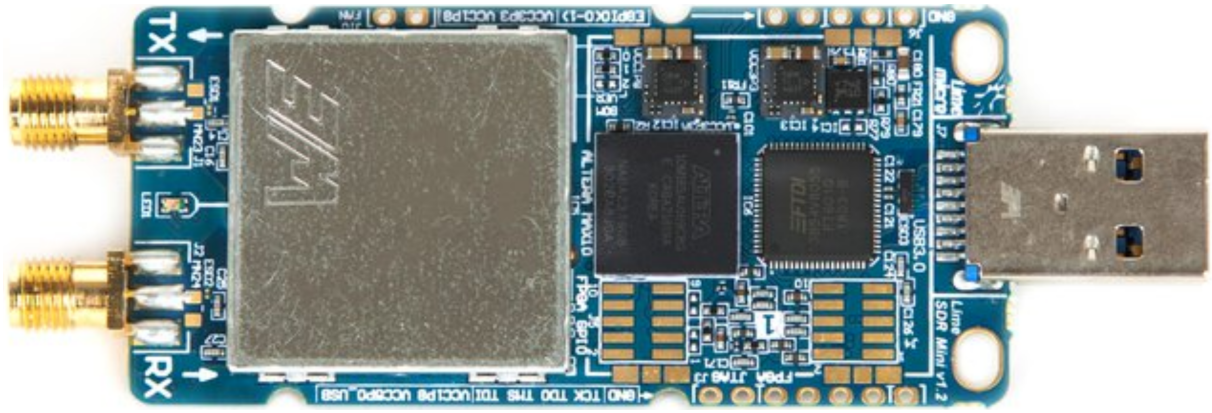

Figure 1.1: The Lime Microsystem LimeSDR-Mini.

ultimate goal is to use the LimeSDR-Mini and a single-board computer to implement a short-range ground-penetrating radar (GPR) to monitor the health of a glacier. To verify if a ground-penetrating radar is possible using the LimeSDR-Mini, a similar but simpler radar system is implemented. A linear frequency-modulated continuous-wave (LFMCW) radar system is implemented to test the capabilities of the LimeSDR board.

1.2 Background

SDRs are extremely versatile because they are programmable and configurable. These capacities have led to research exploring different uses of SDRs. An example is the use of an SDR in a biomedical imaging system [1]. The SDR replaces the costly and bulky vector network analyzer used in the microwave-based medical imaging system. With an SDR, the system successfully detects small targets embedded in human tissues.

Several radar systems have been demonstrated that are based on SDR technology such as weather surveillance radar [2], automotive radar [3], ship radar [4], ground-penetrating radar for landmine detection [5], human movement monitoring radar [6], and multifunctional radar and data communication system [7]. Another group has used the USRP series board attached to an aerial drone for landmine detection [5]. This group used a ground-penetrating radar attached to an UAV (unmanned aerial vehicle).

A majority of these studies have used a Universal Software Radio Peripheral (USRP) SDR [8]. This is a range of commercial software-defined radios designed and sold by Ettus Research

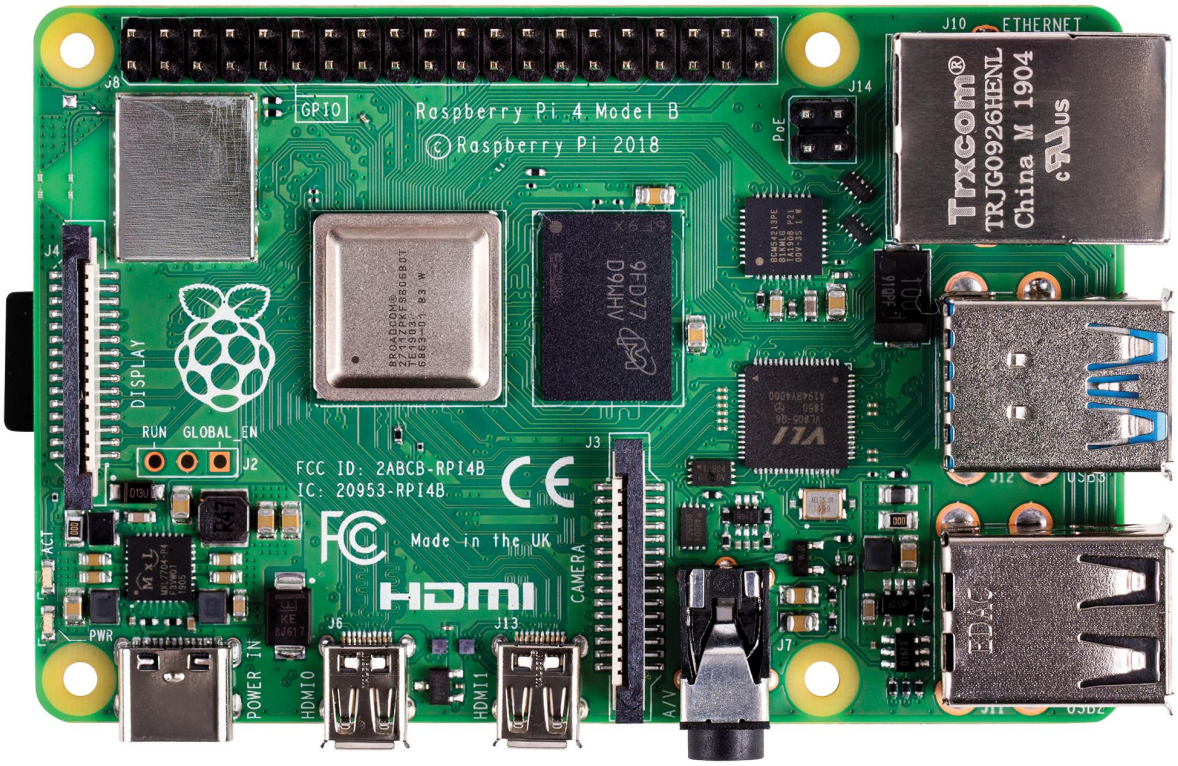

Figure 1.2: Raspberry Pi 4 Model B single-board computer.

and its parent company, National Instruments. The USRP series of software-defined radios are relatively more expensive than the Lime SDRs designed and sold by Lime Microsystems. The cheapest USRP is the USRP B205MINI (one input, one output), which costs about US$850. Other USRP models cost more than US$1,100. In contrast, the LimeSDR-Mini (one input, one output) cost is US$175 while the LimeSDR-USB (two inputs, two outputs) cost is US$315. The lowest cost USRP SDR with two inputs and two outputs is US$1,880, for these reasons, the LimeSDR-Mini was chosen for the radar system. Since the USRP series of SDRs have more products and has been around longer, there is more support and research done with these products. The LimeSDR-Mini (the first batch was produced in 2018) is relatively new and low-cost, but has less capabilities than USRP systems and lacks extensive support. The LimeSDR-Mini is physically smaller and lighter in weight.

Environmental Science professors at Brigham Young University are interested in monitoring the health of a glacier in Switzerland. This can be done by observing the glacier using a ground-penetrating radar mounted on a modern UAV (unmanned aerial vehicle) or drone. My

research in this thesis is centered on creating software tools and starting the designing and building of a ground-penetrating radar using a LimeSDR-Mini, custom antennas, and a Raspberry Pi 4 single-board computer. This is similar to the research done by certain students in Indonesia. Their research looked at designing an LFMCW ground-penetrating radar for concrete inspection using a LimeSDR-Mini [9]. Their research has very limited results. In a simple test, they were able to differentiate a steel reinforcement bar from a concrete surface, but they were unable to produce correct range measurements.

A linear-frequency modulated continuous-wave radar implementation on the LimeSDR is a good test of the capabilities of the hardware and helps develop an understanding of the hardware. Since a ground-penetrating radar also uses chirps, the development of the LFMCW radar overlaps with the development of a GPR. A well-documented open-source complete implementation of an LFMCW radar on the LimeSDR does not currently exist. However, developments in LFMCW radar have been made in other SDR series (primarily USRP device developed by Ettus Research [8] [10] [11]).

1.3 Thesis Statement and Results Summary

The ultimate goal is to design a ground-penetrating radar using the LimeSDR-Mini. Short-range radar is successfully demonstrated using the LimeSDR-Mini. However, consistent detection of a target is not possible due to the instability of the LimeSDR-Mini. The LimeSDR-Mini falls short of operating reliable in a radar system and other SDR boards need to be explored as viable options.

The LimeSDR-Mini can detect targets at a short-range. A test was performed with the LimeSDR-Mini connected to custom antennas and a target placed at various ranges (17 to 34 meters). The LFMCW radar system can detect the target and find the correct range of the target.

Tests (using cables and antennas) show the LimeSDR-Mini is unstable and the transmitter and receiver are not synchronized. Signal processing techniques can mitigate these issues. In addition, a prevalent issue in short-range radars is the signal leakage from the transmitter to the receiver. The instability of the LimeSDR-Mini reduce the effectiveness of signal processing techniques used to eliminate the leakage signal.

1.4 Roadmap

To understand how to implement the LimeSDR-Mini as a radar, the hardware and configuration software is explained in Chapter 2. A simple background of ground-penetrating radar is explored in Chapter 3 and includes why an LFMCW radar is implemented instead of a GPR.

To test the capabilities of the LimeSDR-Mini, an LFMCW radar is implemented in Chapter 4. Section 4.1 explains the theory of an LFMCW radar and is independent of the LimeSDR-Mini. Section 4.2 describes the software tools used to configure the SDR and explains the issues that result from the transmitter and receiver not being synchronized. To overcome the synchronization issue Section 4.3 describes signal processing techniques used to mitigate the problem. With these methods, Section 4.4 describes the test and results of measuring a long length of coaxial cable. This test imitates the use of antennas detecting a target. Section 4.5 describes how the antennas were designed and built. Transmitting and receiving continuously with antennas next to each other result in the received signal being saturated by the signal propagated from the transmitter. This signal is known as the bleed-through signal and is a prevalent issue in short-range radars. The bleed-through signal is considered in further detail in Section 4.6. In Section 4.7 signal processing techniques used to eliminate the bleed-through signal in post-processing are described and tested.

Chapter 5 show the results of detecting targets using antennas. The results are not consistent, but the LimeSDR-Mini was able to perform as a short-range radar. The conclusion in Chapter 6 describes how the LimeSDR-Mini is unsuited to perform as a reliable radar system.

CHAPTER 2. HARDWARE AND CONFIGURATION SOFTWARE

2.1 Hardware

The concept of software-defined radios is not new. Traditionally, the majority of signal processing was done in hardware, but now evolving capabilities of digital electronics make it possible to implement signal processing through software which was only once theoretically possible.

An ideal software-defined radio has a general-purpose processor that handles all of the signal processing, rather than being done in special-purpose electronic circuits (see Diagram A in Figure 2.1). The ideal SDR has a digital-to-analog converter and an analog-to-digital converter connected directly to the processor and the antennas. This gives the processor absolute control and handles all of the signal processing. However, this configuration is not yet possible in practice at high frequencies. As a result, Diagram B in Figure 2.1 is used as the architecture in most SDRs including the LimeSDR-Mini.

The LimeSDR-Mini architecture employs the Altera's Cyclone IV FPGA (field-programmable gate array) in between the processor and the analog and digital converters. This FPGA serves as a FIFO (first in first out) buffer for the samples. In the LimeSDR-Mini, the FPGA also handles the timestamps. The RF Module is the Lime Microsystems transceiver RFIC (radio-frequency in-

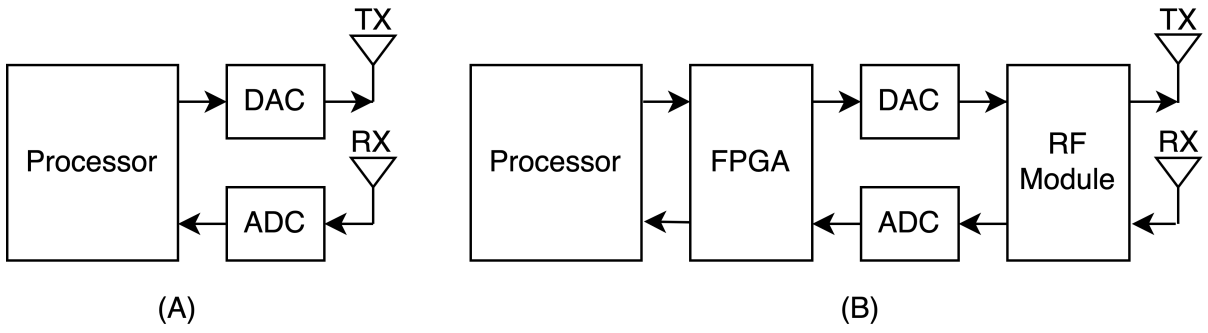

Figure 2.1: Simplified architecture block diagram of an ideal SDR (A) and an typical SDR (B).

Table 2.1: Comparisons in features between the LimeSDR-Mini and LimeSDR-USB.

Lime Microsystem SDR Board	LimeSDR-Mini	LimeSDR-USB
Dimensions	69 mm x 31.4 mm	100 mm x 60 mm
Number of RX Channels	1	2
Number of TX Channels	1	2
Available Radio Frequency Bandwidth	30.72 MHz	61.44 MHz
Minimum Operating Frequency	10 MHz	100 kHz
Maximum Operating Frequency	3.5 GHz	3.8 GHz
FPGA Size	16,000 gates	40,000 gates

tegrated circuit) that is used to transmit and receive radio signals. Anything that happens between the FPGA and the RF Module adds group delay, such as digital up and down conversion chains and buffering. This delay is hard-coded into the Osmocon GSM (Global System for Mobile Communications) stack and is necessary for the hardware to work reliably. This delay impacts the ability to synchronize the timing between the transmitter and receiver. Since radar systems have strict timing requirements this delay needs to be incorporated in signal processing for the radar to work. An understanding of this hardware is needed to understand the capabilities of the software when implementing a radar system.

The LimeSDR-Mini has the same LMS7002M transceiver RFIC as the LimeSDR-USB. In the scope of this project either can be used; however, the LimeSDR-Mini is preferred because of the small form factor. The main difference is that the LimeSDR-Mini has only two channels (a transmit and receive) compared with four channels (two transmit and two receive) on the LimeSDR-USB. The other differences are summarized in Table 2.1. Both boards can transmit at up to 10 dBm (10 mW) and offer an oscillator precision of $\pm$1ppm initially and $\pm$4ppm stably.

The LimeSDR boards can be configured to operate at different parameters. For each channel the gain, carrier frequency, FIFO buffer size, low-pass filter, throughput latency factor, and data type can be set (see Appendix A). In addition, the software can access the status of the FPGA FIFO buffers. This includes the timestamps, data transfer rate, dropped packets, size, and the amount filled.

A single-board computer Raspberry Pi 4 (4 GB RAM model) is used to interface and provide power with the LimeSDR-Mini. They are interconnected via a USB 3.0 port and the

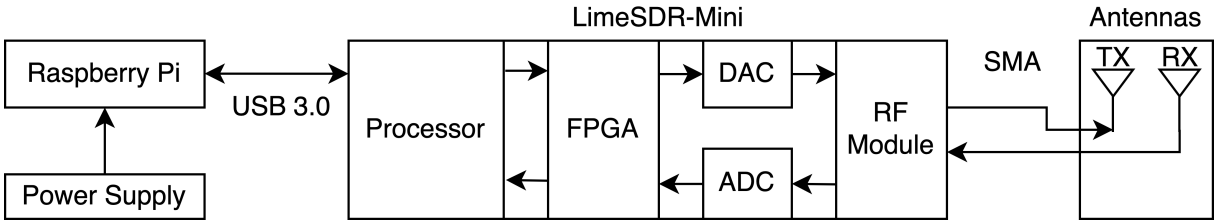

Figure 2.2: Diagram of how all the components are interconnected.

LimeSDR-Mini is connected with the antennas via SMA connectors (see Figure 2.2). The Raspberry Pi is used to communicate and interface with the LimeSDR-Mini. Custom software downloads the RF samples into a binary file for offline processing or to do signal processing and plot the results.

2.2 Configuration and Software

There are multiple software tools to configure the LimeSDR (see links in Table 2.2). A common way to configure multiple varieties of SDRs is to use GNU Radio Companion. GNU Radio Companion is an open-source, free software development toolkit that provides signal processing blocks that can be connected together and to blocks representing hardware (such as the LimeSDR-Mini). To support the LimeSDR, additional libraries are needed to be downloaded. This is one of the most straightforward ways to configure SDRs. However, GNU Radio Companion is mostly used for signal processing and wireless communication protocols.

An alternate software package to configure the LimeSDR is the Soapy Library API. SoapySDR is an open-source generalized API and library for interfacing with most off-the-shelf SDRs. The library is written in C++ and comes with python wrappers. The Soapy Library gives more control of the LimeSDR, but requires more knowledge of SDR hardware since it is a lower level of abstraction than GNU Radio. The Soapy Library was written as a generic tool to program many types of SDRs.

Another alternate is also a pyLMS7002Soapy python library package. The Soapy Library API is essentially a wrapper of the pyLMS7002Soapy library for the LimeSDR. It requires a more extensive understanding of the LMS7002M, Lime's second-generation field programmable RF

Table 2.2: Links to LimeSDR configuration software.

GNU Radio Companion: https://www.gnuradio.org
Soapy SDR: http://pothosware.com
pylms7002m Library: https://myriadrf.org/projects/software/pylms7002m
Lime Suite: hhttps://myriadrf.org/projects/software/lime-suite

transceiver integrated circuit. This library is strictly for use of the LimeSDRs. The purpose of this library is for fast prototyping and algorithm development.

The C++ Lime Suite library package has the most control over the LimeSDR. Since the library is specific to the LimeSDR hardware, it cannot be ported to other SDRs, unlike SoapySDR and GNU Radio. Using The Lime Suite library requires the most understanding of the LimeSDR hardware, but provides the most transparency to how software controls the hardware. The Lime Suite library provides a C-style API and is well documented. This C++ library package provides more control of the LimeSDR than the other abstracted options. The Lime Suite library is used in this thesis since it provides the most control and transparency in how the hardware functions.

CHAPTER 3. GROUND-PENETRATING RADAR

3.1 Ground-Penetrating Radar

Brigham Young University geology professors are interested in monitoring the health of a glacier in Switzerland. The small form factor of the LimeSDR-Mini and Raspberry Pi 4 allows it to fit on an air drone quadcopter. Since SDRs are configurable the radar system can be easily changed from an LFMCW radar to a ground-penetrating radar by running different software. The development of the LFMCW radar and GPR have a significant overlap. Both use the same LimeSDR-Mini, single-board computer, software library packages, and antennas. Both types of radars also use linear-frequency modulated chirps, thus the development of the LFMCW radar makes for a simple transition to a GPR.

Ground-penetrating radar is a geophysical technique that provides a non-invasive method to probe the ground. GPR uses electromagnetic waves to penetrate the earth's surface to detect buried objects or differentiate between soil layers. Since water and ice have different dielectric properties, GPR can be used to find layers of water beneath ice. Additionally, radars have been used to do differentiate between new and old snow [12].

The health of glaciers can be monitored by looking at the glacier mass balance, the difference between accumulation and ablation. Climate change causes fluctuations in the surface mass balance of a glacier. Using radars to measure the health of glaciers is not a new practice and research in the field have been ongoing for over half a century. The research field of radio-glaciology is the study of glaciers, ice sheets, ice caps, and icy moons using ice-penetrating radar. This employs the same geophysical method as a ground-penetration radar and typically operates at frequencies in the MF, HF, VHF, and UHF portions of the radio spectrum [13].

The health of glaciers is effectively measured with radars, due to the characteristics of ice. The conductivity, the imaginary part of the permittivity, and the dielectric absorption of ice are small. This results in low loss tangent, skin depth, and attenuation values [14]. In addition, it

allows echoes from the base of the ice sheet to be detected through ice thicknesses greater than 4 km [15].

The carrier frequency of a ground-penetrating radar is chosen to match the application and has trade-offs. A GPR operating at lower frequencies has a greater penetration depth, while a GPR operating at higher frequencies provides a greater resolution to detect smaller features. The trade-off is between resolution and penetration depth. A GPR with low frequencies can potentially penetrate through hundreds of meters of ice.

The LimeSDR-Mini and LimeSDR-USB are able to operate from the HF band to the UHF band. The LimeSDR-USB can go as low as the MF band. These carrier frequencies are effective for GPR in ice. The theory and full implementation of the GPR are beyond the scope of this paper.

CHAPTER 4. LINEAR FREQUENCY MODULATED CONTINUOUS WAVE RADAR

4.1 LFMCW Radar Theory

A linear frequency-modulated continuous-wave (LFMCW) radar is implemented on the LimeSDR-Mini to test its functionality. Continuous-wave implies that the radar is constantly transmitting and receiving. In bistatic radar there are two antennas; a transmitter and a receiver.

The five main components and processing steps in an LFMCW Radar described below. A flow diagram of the components is shown in Figure 4.1.

1. **Synthesize an up-chirp:** The first step is for the LimeSDR to synthesize an up-chirp. An "up-chirp" is a signal that increases linearly in frequency over time (see Equation 4.3). This is the "linear frequency-modulated" in an LFMCW radar. The parameter f_0 is the starting frequency of the chirp and the parameter f_1 is the end frequency of the chirp. The parameter B is the bandwidth of the chirp. This is the frequency range that is swept.

$$B = f_1 - f_0. \tag{4.1}$$

Another variable is the length of the chirp is in seconds. The time length of a single chirp is the variable T_c. Since this is a continuous-wave radar, once a chirp is finished, another identical chirp immediately follows. Additionally, the slope of the linear frequency increase over time of the chirp is an important characteristic in an LFMCW radar. The equation

$$S = \frac{B}{T_c} \tag{4.2}$$

gives frequency slope S in $\frac{\text{Hz}}{\text{s}}$of the chirp. This is used in further signal processing.

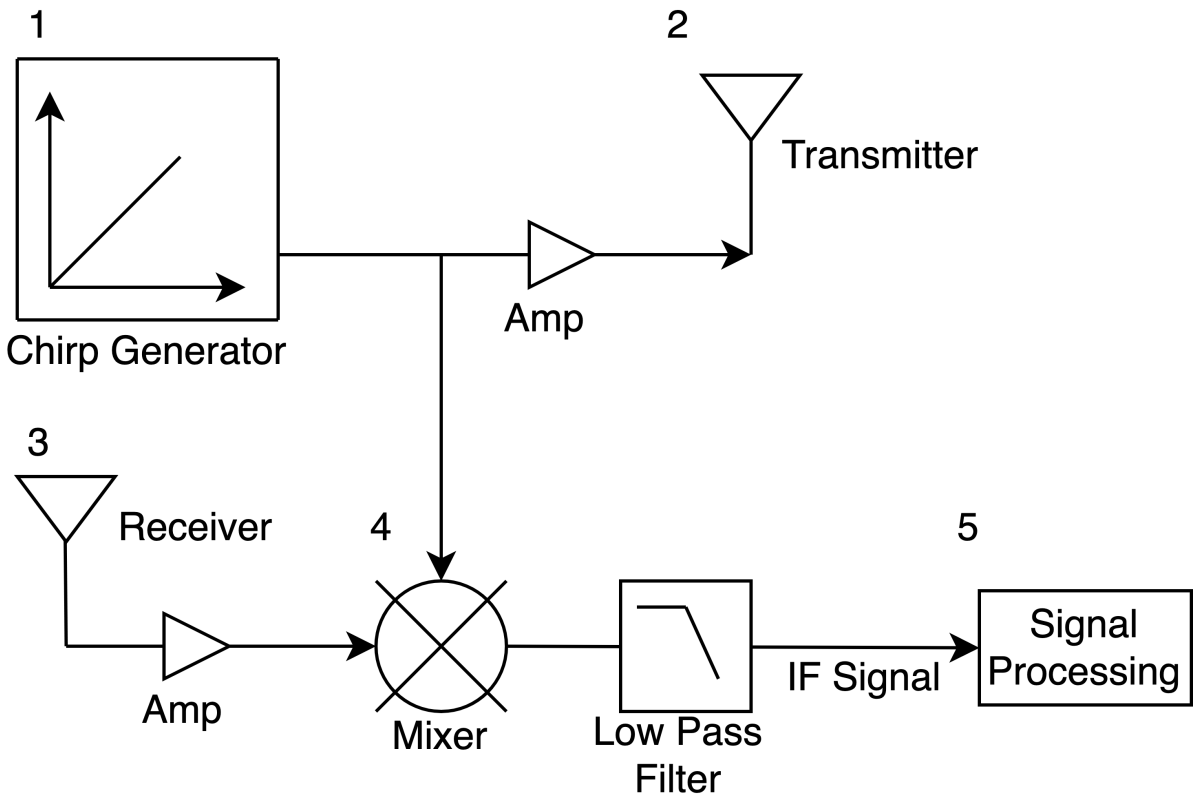

Figure 4.1: The major components of an LFMCW radar.

In a linear-frequency chirp the instantaneous frequency $f(t)$ varies linearly with time, i.e

$$f(t) = S\,t + f_0, \tag{4.3}$$

where $0 \leq t \leq T_c$ and t is in seconds. This frequency waveform is repeated for each chirp.

For any oscillating signal, the integral of the frequency function is the time-domain function for the phase. The derivative of the phase is the angular frequency; $\phi'(t) = 2\pi f(t)$. For the linear chirp, this results in:

$$\begin{aligned}\phi(t) &= \phi_0 + \int_0^t f(\tau)d\tau \\ &= \phi_0 + \int_0^t (S\tau + f_0)d\tau \\ &= \phi_0 + 2\pi\left(\frac{S}{2}\,t^2 + f_0\,t\right), \end{aligned} \tag{4.4}$$

where ϕ_0 is the initial phase [16]. The corresponding time-domain function for a linear up-chirp is the cosine of the phase in radians. This results in the function

$$x(t) = \cos\left(2\pi\left[\frac{S}{2}\,t^2 + f_0\,t\right] + \phi_0\right). \tag{4.5}$$

In contrast, the function of a time-domain complex up-chirp is

$$\begin{aligned} x(t) &= \exp\left(j\left[2\pi\left\{\frac{S}{2}\,t^2 + f_0\,t\right\} + \phi_0\right]\right) \\ &= \cos\left(2\pi\left[\frac{S}{2}\,t^2 + f_0\,t\right] + \phi_0\right) + j\sin\left(2\pi\left[\frac{S}{2}\,t^2 + f_0\,t\right] + \phi_0\right), \end{aligned} \tag{4.6}$$

where j is the imaginary number. In this paper, the complex up-chirp signal is used and the initial phase ϕ_0 is zero. Synthesizing the up-chirp is done in software on the single-board computer using the above equation.

2. **Transmit chirp:** The second step for an LFMCW radar is to send the complex up-chirp (Equation 4.6) to the transmit antenna. The transmit signal is amplified using the LimeSDR-Mini hardware. The gain of the amplifier is configured in software. The up-chirp is transmitted towards and reflected from the target.

3. **Receive echoed signal:** The third step of the LFMCW radar is to receive the signal that echoes back from the target. This signal is similar to the transmitted signal but is delayed in the time based on how far the target is from the radar (see Figure 4.2). The delay τ is the time of flight it takes the electromagnetic signal to go to the target and back. If R is the distance from the antenna to the target being detected, then the signal needs to travel there and back, a

total distance of $2R$. Electromagnetic signals travel near the speed of light through air. Thus,

$$\tau = \frac{2\,R}{c}, \tag{4.7}$$

where c is the speed of light. The receive signal is amplified by the LimeSDR-Mini hardware with the gain configured in software.

4. **Mixer:** The next step is to mix the transmit and receive signals. Mixing is done by multiplying the signals in the time domain, which is equivalent to convolving the signals in the frequency domain. When using two real signals with different frequencies the mixer produces a real signal with a frequency of $|f_1 - f_2|$ and $|f_1 + f_2|$. The new frequencies produced are called intermediate frequencies. In an LFMCW receiver, the low pass filter removes the sum term, leaving only the difference form.

   In the case of the LFMCW radar in this thesis, a complex signal is used. Mixing complex signals is the same as mixing real signals, except only a single frequency is produced. By mixing the complex transmit and the complex conjugate of the receive signal, the mixer produces the difference of their frequencies. The conjugation causes the mixer to subtract the receive frequency from the transmit frequency, resulting in a positive f_{IF} frequency.

   The transmit signal and receive signal are similar, except the receive signal is delayed by τ. The following equation shows how the mixed signal time-domain function $f_{\text{IF}}(t)$ is produced by multiplying the transmit signal by the complex conjugate of the receive signal.

$$\begin{aligned}
f_{\text{IF}}(t) &= \exp\left(j2\pi\left[\frac{S}{2}t^2 + f_0 t\right]\right)\exp\left(-j2\pi\left[\frac{S}{2}\{t-\tau\}^2 + f_0\{t-\tau\}\right]\right) \\
&= \exp\left(j2\pi\left[\frac{S}{2}t^2 + f_0 t - \frac{S}{2}\{t-\tau\}^2 - f_0\{t-\tau\}\right]\right) \\
&= \exp\left(j2\pi\left[\frac{S}{2}t^2 + f_0 t - \frac{S}{2}\{t^2 - 2\tau t + \tau^2\} - f_0 t + f_0\tau\right]\right) \\
&= \exp\left(j2\pi\left[S\tau t - \frac{S}{2}\tau^2 + f_0\tau\right]\right) \\
&= \exp(j2\pi S\tau t)\exp\left(j2\pi\left[f_0\tau - \frac{S}{2}\tau^2\right]\right) \\
&= \exp(j[2\pi S\tau t + \phi]),
\end{aligned} \tag{4.8}$$

where ϕ is the phase offset $j2\pi\left(f_0\tau - \frac{S}{2}\tau^2\right)$. This equation is valid for $\tau \leq t \leq T_c$, because the useful data is produced when the transmit signal has a higher frequency than the receive signal.

The f_{IF} signal can be inferred from Figure 4.2. A simpler way to find the f_{IF} frequency is to ignore the phase. Since the chirps are linear, subtracting the two chirps result in a constant difference frequency. The transmit instantaneous frequency is represented by Equation 4.3, $f(t)$. The receive instantaneous frequency is also represented by Equation 4.3, but is delayed by τ, $f(t-\tau)$. The equation

$$\begin{aligned} f_{\text{IF}} &= f(t) - f(t-\tau) \\ &= S\,t + f_0 - (S(t-\tau) + f_0) \\ &= S\tau \end{aligned} \tag{4.9}$$

shows a simpler way to find the f_{IF} frequency, but does not derive the phase offset.

The f_{IF} signal can be related to the range of the object detected by substituting τ in Equation 4.9 with Equation 4.7 to produce Equation 4.10. Now instead of requiring the variable τ, the f_{IF} signal can be related to range by

$$R = \frac{f_{\text{IF}}\,c}{2\,S} = \frac{\tau\,c}{2}. \tag{4.10}$$

5. **Signal Processing:** The last step is to do further signal processing on the mixed signal and find the f_{IF} signal to calculate the range. Typically, a discrete Fourier transform (DFT) is performed on the magnitude of the signal. This converts the mixed signal from the time domain to the frequency domain. The magnitude is given by p(t) = $\sqrt{\text{real}(y(t))^2 + \text{imag}(y(t))^2}$. The power spectrum density function is simply the DFT of p(t). The power spectrum units are in dB of an unknown voltage-squared value. For reference, the LimeSDR-Mini is capable transmitting up to 10 dBm.

   Further signal processing techniques can also be implemented. For example, having multiple peaks in the DFT mixed signal means having multiple targets detected. The f_{IF} signal tones can be related to the range of multiple objects with Equation 4.10. Doppler shifts resulting

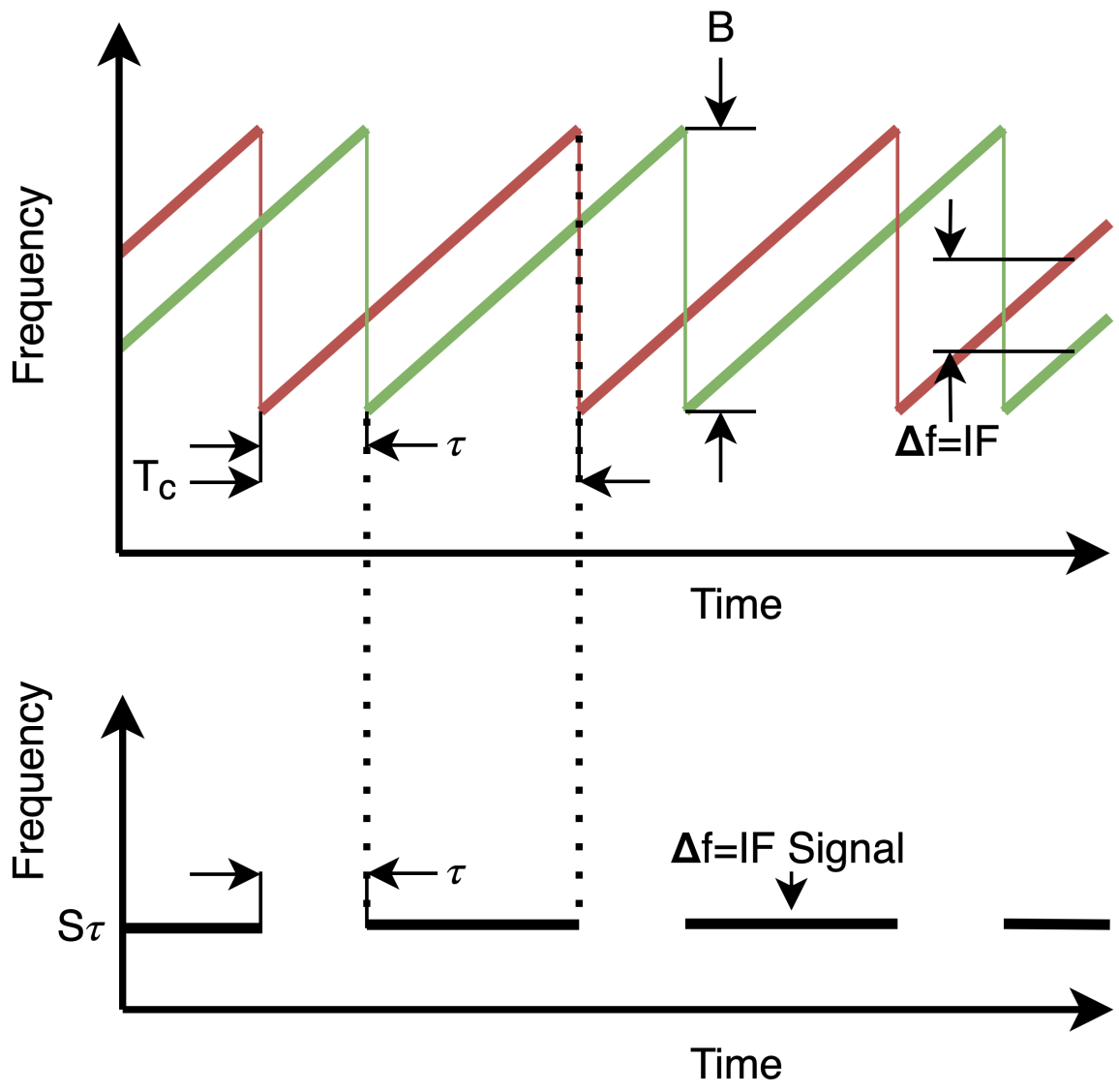

Figure 4.2: The top axes show the LFMCW transmit chirp (red) and receive chirp (green). The mixer produces the f_{IF} signal as shown in the bottom axes.

from moving targets can also infer the velocity of the targets. Since the ultimate goal of developing the LFMCW radar is for a GPR where the target and clutter is stationary, complex analysis of the LFMCW radar is not pursued.

There are some important signal processing techniques to improve the accuracy and SNR of the f_{IF} signal. Performing a discrete Fourier transform on a signal divides the signal into small discrete ranges of frequencies called bins. These bins are called frequency bins, and contain the

net power of the frequencies inside the bins. To increase the accuracy of the LFMCW radar, the width of the frequency bins needs to be minimized. The width of the frequency bin Δf is found with the equation

$$\Delta f = \frac{f_\mathrm{s}}{N}, \tag{4.11}$$

where f_s is the sampling frequency and the number of samples in the DFT is N. To minimize the frequency bin width, f_s needs to be minimized and N needs to be maximized. Δf can only be minimized to a certain point by minimizing f_s because of the Nyquist–Shannon sampling theorem. Thus using more samples by increasing the acquisition time to perform a DFT is more practical.

To convert the frequency bins into range bins ΔR, Equation 4.10 is applied by replacing f_IF with Δf. This produces the equation

$$\Delta R = \frac{\Delta f \; c}{2 \; S}. \tag{4.12}$$

This is used to plot the magnitude versus range of the mixed signal. With range as the independent variable, it is easier to interpret the data if targets are detected.

To accurately measure the range of a target, the range bins defined in Equation 4.12 need to have a high enough resolution to be practical. Combining Equation 4.11 with 4.12 and letting the number of samples $N = f_\mathrm{s} \; T_c \; k$ results in the equation

$$\Delta R = \frac{\Delta f \; c}{2 \; S} = \frac{\frac{f_\mathrm{s}}{f_\mathrm{s} \; T_c \; k} c}{2 \; \frac{B}{T_c}} = \frac{c}{2 \; k \; B}, \tag{4.13}$$

where k is a ratio of how many samples are in the DFT and the number of samples in a single chirp. Note that N needs to be a positive integer. ΔR is also commonly referred to as the range resolution.

This equation shows that to improve the range resolution ΔR, the chirp bandwidth B is maximized, and the number of samples in the DFT is increased. The bandwidth B is limited by the sampling frequency f_s, which is limited by the hardware. To minimize ΔR, the number of samples the DFT is increased by selecting a single receive chirp interval and zero-padding the end. The addition of zeros to the end of the time-domain waveform does not improve the underlying frequency resolution but leads to an interpolated DFT result, which can produce a higher display resolution. Since zero-padding essentially introduces a rectangle function to the end of the time-domain waveform, the DFT produces sinc-like functions.

To detect a return signal with low power, the signal power needs to be strong enough to discriminate against the noise. Noise is a general term for unwanted (and, in general, unknown) modifications that a signal may suffer during capture, storage, transmission, processing, or conversion [17]. In order to increase the SNR (signal-to-noise) ratio of the f_{IF}, multiple receive signals are averaged together and then mixed with the transmit chirp. This technique lowers the noise floor. In a test of averaging 100 chirps together, the noise floor dropped by 6 dB. The combination of zero-padding and averaging the receive chirps together results in further resolution and SNR of the f_{IF} signal. This makes the radar more accurate and dependable.

The maximum range R_{max} is constrained from both the signal attenuation and signal processing capabilities. The further the target is from the radar the less power is returned from the echo. The power return of a signal drops quadratically according to R^{-4} in the radar range equation [18]. If the target is too far then the return signal can not be discriminated from the noise. The maximum range is also limited by the ADC sampling frequency f_s. This results in

$$R_{\text{max}} = \frac{f_s \, c}{2 \, S}. \tag{4.14}$$

This equation is directly related to Equation 4.10. Since the radar developed is a short-range radar, R_{max} is not overly important.

4.2 LimeSDR-Mini Configuration

The LFMCW radar implementation was first tested using a 150-meter length of coaxial cable that is connected from the transmitter to the receiver. This mimics the behavior of a radar transmitting a signal: having the signal echo from a target, and then receiving the echo. This simplifies the testing by eliminating multi-path and interference. The signal is clean, making it easier to debug. It is important to note that in this set up, the scalar 2 is no longer in Equation 4.7 and in Equation 4.10. This is because the signal is going through the coaxial cable directly from the transmitter to the receiver (distance R) and is not echoing off a target (distance $2R$). Also, the speed of signal propagation is slower since electromagnetic waves propagate slower through a wire medium (roughly $\frac{2}{3}c$) than in free space or air.

The first method tested for implementing an LFMCW radar was using GNU Radio Companion (GRC). GNU Radio Companion is a free software development toolkit that provides signal processing blocks to implement software-defined radios and signal-processing systems. Simulating an LFMCW radar in GRC was implemented and worked as expected. However, when the LimeSDR-Mini is used instead of a simulation, the intermediate frequency f_{IF} is erroneously high and varies over time. For initial testing, a 10 cm meter cable was used to connect the transmitter and receiver together. With the transmitter and receiver shorted together, the transmit signal and receive signal occur at nearly the same time. This resulted in a low f_{IF} signal frequency. However, the results showed the f_{IF} signal erroneously high and variable. The f_{IF} signal varied greatly as the system ran, proving unusable data. To mitigate this problem, the software was moved from a virtual machine to the Raspberry Pi single-board computer. This improved the accuracy and jitter of the f_{IF} signal, but the data was still far from usable. Since GNU radio was created to be a generic tool for signal processing, it is difficult to use it to debug the specific problems within the LimeSDR hardware. In summary GNU radio companion works well when there are not strict timing requirements for transmitting and receiving, but since radar requires precise timing, greater control of the LimeSDR is needed.

Because there is a lack of transparency in GNU Radio software, the decision was made to move to another method of programming the LimeSDR. The pothosware SoapySDR Library was found to be a high enough level abstraction that in depth knowledge of the hardware is not needed to be known, but low enough that there was more control of the hardware. The SopaySDR is able to provide timestamps for when to transmit a signal and receive a signal. This method gave better control to critical timing restraints of a radar system. However, the results still produced an erroneous f_{IF} signal. However, unlike the results from GNU Radio, the f_{IF} signal is consistently the same. This means that the delay should be taken into account in signal processing. SoapySDR is an open-source generalized API and library for interfacing with most off-the-shelf SDRs. It is not specific to the LimeSDR hardware, making it hard to understand how the software interfaces with the LimeSDR. This is because some functions calls do not work specifically with the LimeSDR, but works with other types of SDRs. Although the SoapySDR library works sufficiently for producing LFMCW radar results, other ways of configuring the LimeSDR were examined to determine if the transmitter and receiver could be better synchronized.

The option of using the Myriad-RF pyLMS7002Soapy library was also explored. The purpose of this library is for fast prototyping and algorithm development. This library is more specific to the LimeSDR, but ultimately was not used since there is little documentation on how to use the library. Since the SoapySDR library is essentially a wrapper of the pyLMS7002Soapy library, the SoapySDR is better suited for quick development.

I found the best option for configuring the LimeSDR is the Lime Suite Library. It was created specifically for the LimeSDR. This means the code written with the Lime Suite library is not portable to other types of SDRs, unlike the code made from GNU Radio or SoapySDR. The Lime Suite API is well documented and some example code of simple transmits and receives is found in the Lime Suite repositories. Using this library requires an understanding of the LimeSDR architecture. The API includes functions that give more control over the LimeSDR hardware. In addition, the API gives greater control of timestamps and provides the status of the FPGA FIFO buffers used to transmit and receive. Explanation and references to the API and further documentation of the nuances of programming the LimeSDR can be found in Appendix A.

4.3 Synchronization of the Transmitter and Receiver

A key issue to overcome is the synchronization of the transmitter and the receiver. Unfortunately, configuring the LimeSDR-Mini with the Lime Suite library using software synchronized timestamps does not synchronize the transmitter and receiver in hardware. Ultimately, the hardware does not support perfect synchronization of the transmitter and receiver. The FPGA in the LimeSDR Mini handles the timestamps. Anything that happens between the RF module and the FPGA adds group delay, such as digital up and down conversion chains, and buffering. This delay is hard coded for each type of SDR hardware in the Osmocon GSM (Global System for Mobile Communications) stack and is required for the hardware to work reliably, making the delay necessary. This delay is referred to as τ_{FPGA} (in seconds). τ_{FPGA} in this paper is defined as the difference between the literal transmit and receive time when the timestamps are equivalent in software. If the transmit timestamp in software is x, the hardware transmits the data at time x. However, if in the software the receiver is also synced to transmit at time x, then the hardware receives data at time $x + \tau_{\text{FPGA}}$. Fortunately, the delay τ_{FPGA} is consistent once the software is running. τ_{FPGA} depends on the parameters set in the LimeSDR; filter configuration, sampling frequency, gain, bandwidth,

etc. It is thus possible to measure this delay and incorporate it into signal processing, effectively eliminating it. The delay τ_{FPGA} measured in the LimeSDR-Mini is anywhere between $2\mu s$ and $80\mu s$

To make the LimeSDR a reliable radar, the delay τ_{FPGA} needs to be known. Once it is known, it can be incorporated into signal processing. The delay τ_{FPGA} can be measured by shorting the receiver and transmitter together with a short coaxial cable. The receive signal (up-chirp) is delayed in time by τ_{FPGA}. Mixing the delayed receive signal with the transmit signal produces the constant intermediate frequency f_{FPGA}. The time delay τ_{FPGA} and the intermediate frequency f_{FPGA} are directly related by Equation 4.9. This same method of finding τ_{FPGA} is applied to the radar when using antennas. In normal operations, the antennas are next to each other so that the signal propagates directly from the transmitter to the receiver without echoing from any targets (called bleed-through signal) and this signal is mixed with the transmit signal is used to produce f_{FPGA}.

The time it takes for the signal to propagate from the transmit antenna and echo back from the target is represented as τ_{target}. There is also the inherent delay τ_{FPGA} in the LimeSDR-Mini. Thus the total time from the transmit timestamp to when the chirp is recorded by the LimeSDR-Mini is the sum of these delays. The total time delay is

$$\tau_{\text{IF}} = \tau_{\text{target}} + \tau_{\text{FPGA}}. \tag{4.15}$$

The intermediate frequency f_{IF} results from mixing the receive signal (delayed by τ_{IF}) with the transmit signal. Modifying Equation 4.9 results in

$$\begin{aligned} f_{\text{IF}} &= S\,\tau_{\text{IF}} \\ &= S\,(\tau_{\text{target}} + \tau_{\text{FPGA}}) \\ &= f_{\text{target}} + f_{\text{FPGA}}, \end{aligned} \tag{4.16}$$

where f_{target} is the intermediate frequency that would be produced from mixing if the LimeSDR had no synchronization issues. Equation 4.16 shows the intermediate frequency f_{IF} is the sum of the intermediate frequencies f_{target} and f_{FPGA}. Thus Equation 4.10 is modified by Equation 4.16 to

be

$$\begin{aligned} R_{\text{target}} &= \frac{f_{\text{target}}\, c}{2\, S} \\ &= \frac{(f_{\text{IF}} - f_{\text{FPGA}})\, c}{2\, S}, \end{aligned} \tag{4.17}$$

where R_{target} is the range of the target.

4.4 Coaxial Cable Test Results

When measuring the length of the 150-meter coaxial cable the f_{IF} measured is a result of the time it takes the signal to propagate through the cable τ_{cable} and the FPGA time delay τ_{FPGA}. Note that an electromagnetic wave propagates roughly $\frac{2}{3}$ the speed of light through a cable and that the distance the signal propagates is the length of the cable, unlike a radar where it goes to a target and back. Equation 4.17 is modified to be

$$R_{\text{cable}} = \frac{(f_{\text{IF}} - f_{\text{FPGA}})\, \frac{2}{3} c}{S}, \tag{4.18}$$

where R_{cable} is the length of the coaxial cable.

The LFMCW Radar software developed using the Lime Suite library successfully measured the length of the 150-meter coaxial cable. The first step was to measure f_{FPGA} which is the intermediate frequency signal when the transmitter and receiver are shorted together (see Figure 4.3). Then the f_{IF} signal is found when the 150-meter coaxial cable is interconnected to the transmitter and receiver (see Figure 4.4). Using Equation 4.18, the length of the 150-meter coaxial cable is measured and compared with the known length. Table 4.1 and Table 4.2 shows the results with different parameters are used in the LFMCW radar. The tables also show how the FPGA time delay τ_{FPGA} varies. While it is true that τ_{FPGA} varies based on the parameters set in the LimeSDR-Mini, it is also true that τ_{FPGA} varies by running the same executable multiple times. This makes it difficult to know the precise τ_{FPGA}. Essentially, it is fixed during each program run, but varies between runs.

The FPGA time delay τ_{FPGA} varies from run to run with the same executable. Running many instances, Figure 4.5 shows a uniform distribution of four discrete times. The distribution

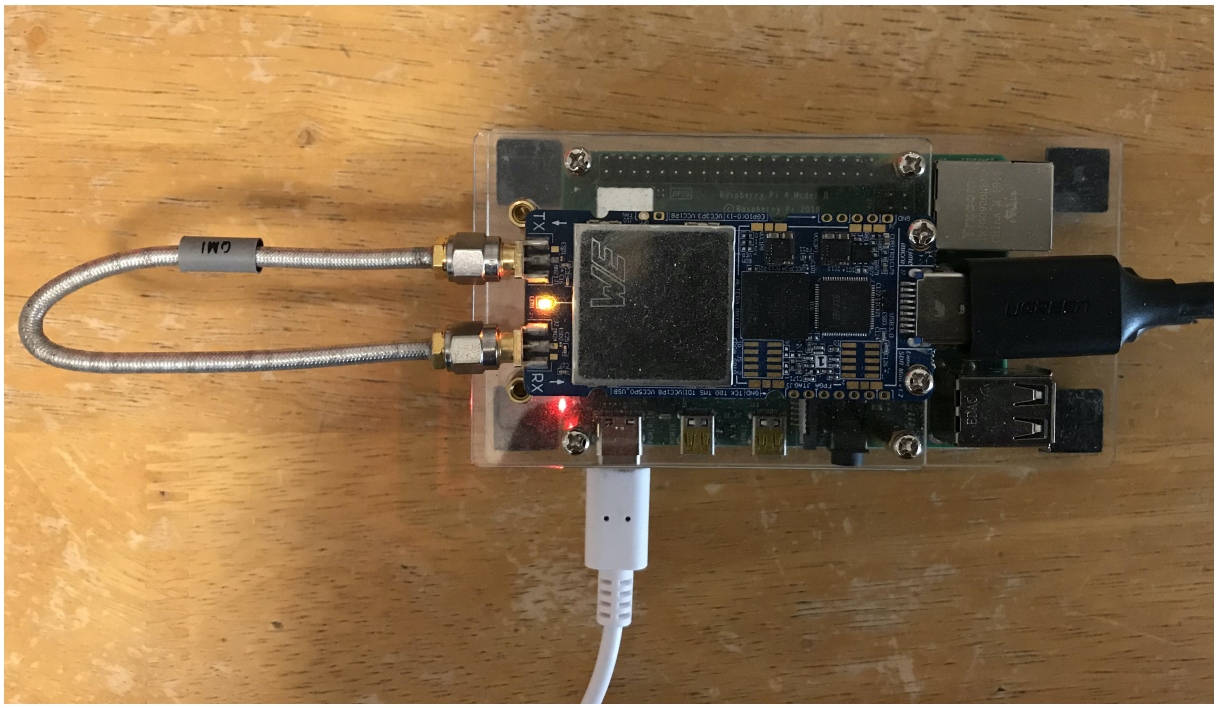

Figure 4.3: Shorting the transmitter and receiver together to find τ_{FPGA} and f_{FPGA}.

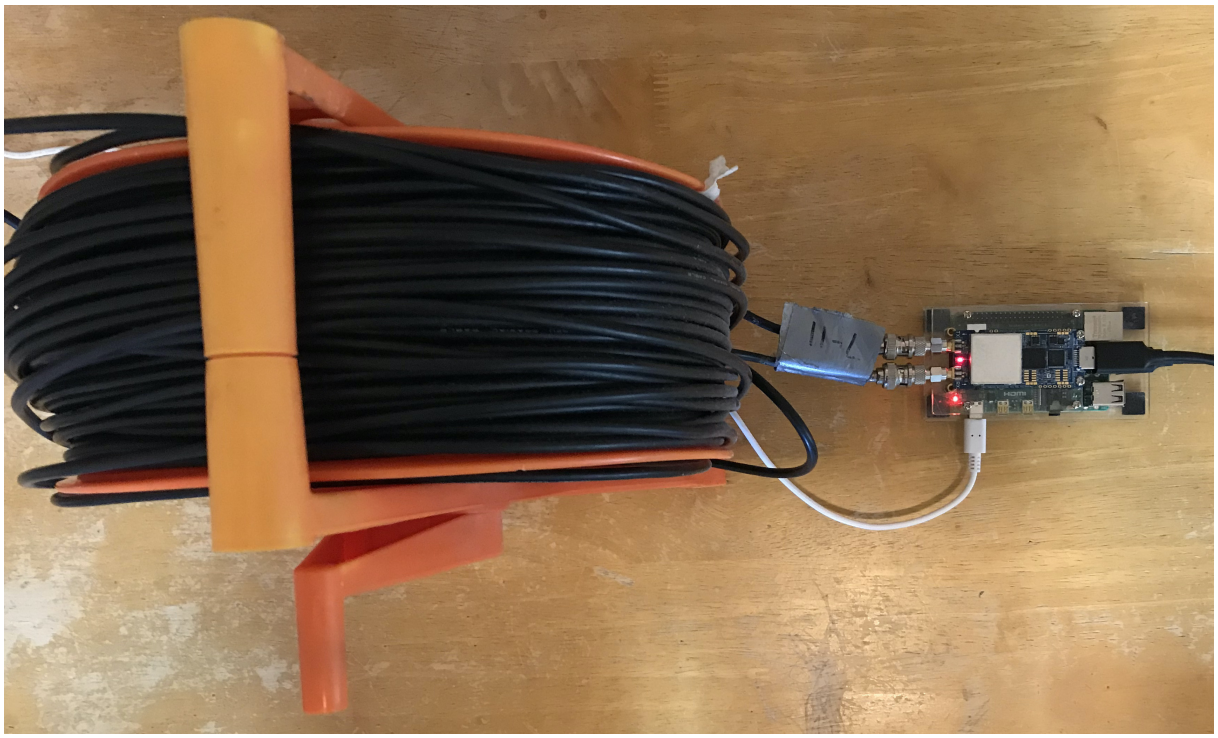

Figure 4.4: 150-meter coaxial cable connected to the transmitter and receiver.

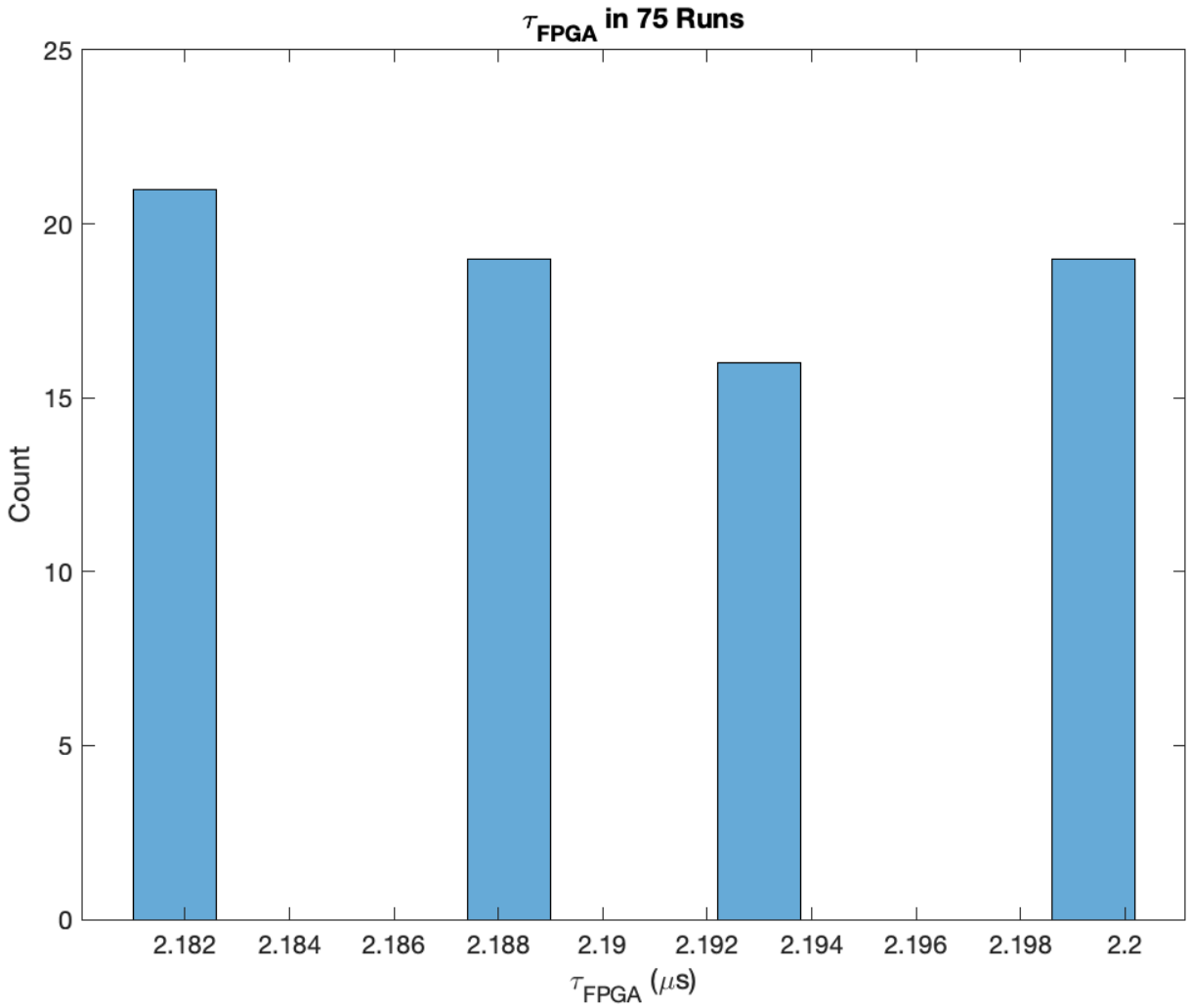

Figure 4.5: Distribution of measured τ_{FPGA} running a single executable multiple times.

of the time delay τ_{FPGA} was found using the sampling frequency f_s of 36 MHz, bandwidth B of 16 MHz, and a chirp length T_c of 100 μs. The maximum difference betweeb the variations of τ_{FPGA} in this example is 18.7 ns, which corresponds to 233.59 mm in distance. These variations of running the same executable are negligible in this test to measure the length of the coaxial cable.

Table 4.1 shows results of changing the bandwidth B of the chirp while other LFMCW variables remain constant. N is defined at the number of samples used in the DFT, where $N = f_{\text{s}}\, T_c\, k$ and k is a positive number. The DFT is taken of the average receive signal and is zero-padded. The results show that the length of the 150-meter cable is measured accurately enough to conclude that the radar is working. There is some variation of the range result due to the range resolution. There appears to be no clear correlation of how τ_{FPGA} varies with the changing bandwidth. Table

Table 4.1: Measuring 150-meter coaxial cable with different chirp bandwidths.

F_s	B	T_c	k	ΔR	f_{FPGA}	τ_{FPGA}	f_{IF}	R
36MHz	2MHz	100μs	20	5m	50kHz	2.50μs	65 kHz	149.90m
36MHz	4MHz	100μs	20	2.5m	94kHz	2.35μs	125kHz	154.89m
36MHz	8MHz	100μs	20	1.25m	108.5kHz	2.56μs	241kHz	151.15m
36MHz	16MHz	100μs	20	0.625m	351kHz	2.19μs	471.5kHz	150.52m

Table 4.2: Measuring 150-meter coaxial cable with different sampling frequencies.

F_s	B	T_c	k	ΔR	f_{FPGA}	τ_{FPGA}	f_{IF}	R
24MHz	8MHz	100μs	20	1.25m	262.5kHz	3.28μs	324kHz	153.64m
30MHz	8MHz	100μs	20	1.25m	235kHz	2.94μs	295.5kHz	151.15m
36MHz	8MHz	100μs	20	1.25m	180.5kHz	2.26μs	242kHz	153.64m

4.2 shows how τ_{FPGA} changes with different sampling frequencies. The sampling frequency and τ_{FPGA} appear to be inversely proportional.

Figure 4.6 shows the DFT of the mixed signal produced with the 150-meter coaxial cable. The parameters used are found in the last row in Table 4.1. Figure 4.6 shows the largest peak at f_{IF}. The second lower peak on the left side of the figure is a result of the mixer when the transmitting chirp has begun a chirp, but an echo of a previous chirp is still being received. Looking at Figure 4.2, this happens when the green receive chirp is above the red transmit chirp. This happens at the beginning of the transmit chirp where $0 \leq t \leq \tau$. The mixer subtracts the lower frequency by the higher frequency, resulting in a negative frequency. This peak can be ignored and even zeroed out.

Figure 4.7 shows the same signal as Figure 4.6, but displays the main peak zoomed in and the x-axis is changed to display the range. This is done by using Equation 4.18, where 0 meters corresponds with the same data point as the measured f_{FPGA}. The sinc behavior is due to zero-padding. Figure 4.7 shows that the length of the coaxial cable is about 150-meters. The results show fairly high precision. The accuracy may be off, by the assumption of the speed of an electromagnetic wave traveling through a wire medium. Note that the LFMCW Radar software accurately measures the length of the 150-meter coaxial cable.

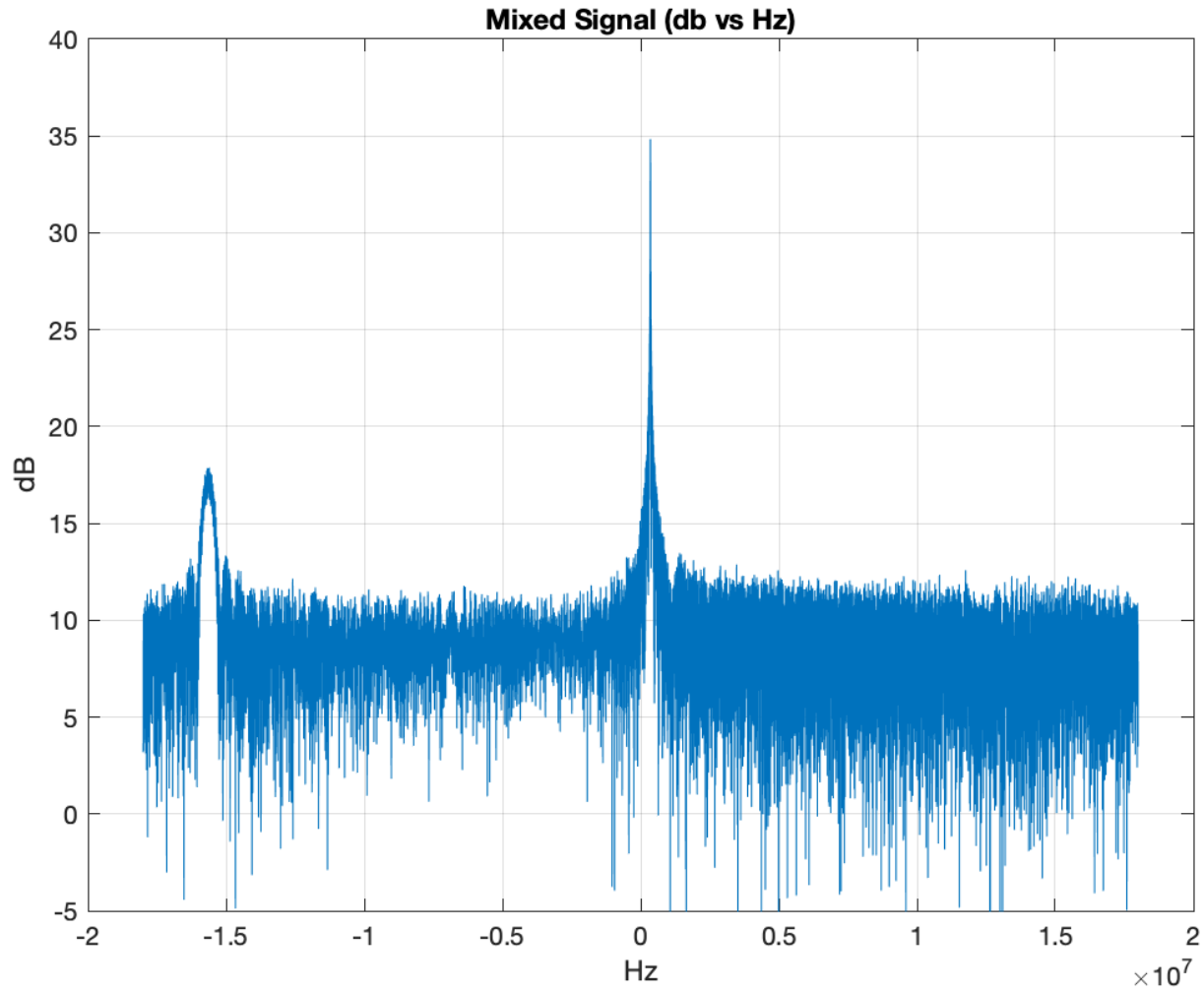

Figure 4.6: DFT of the mixed signal with the 150-meter coaxial cable. The peak to the left can be ignored since it is an artifact of when the transmit chirp starts, but the previous chirp is still being received. The peak near 0 Hz is of primarily interest.

4.5 LFMCW Radar Antenna

To measure the range of targets using the LFMCW radar, custom antennas were built for the receiver and transmitter. The antennas are designed to be compact enough to mount on an air drone and to be efficient over a wide bandwidth.

The antenna efficiency (radiation efficiency) ξ measures how effective an antenna is at transmitting the power delivered to it. The antenna efficiency is

$$\xi = \frac{P_{\text{rad}}}{P_{\text{t}}}, \tag{4.19}$$

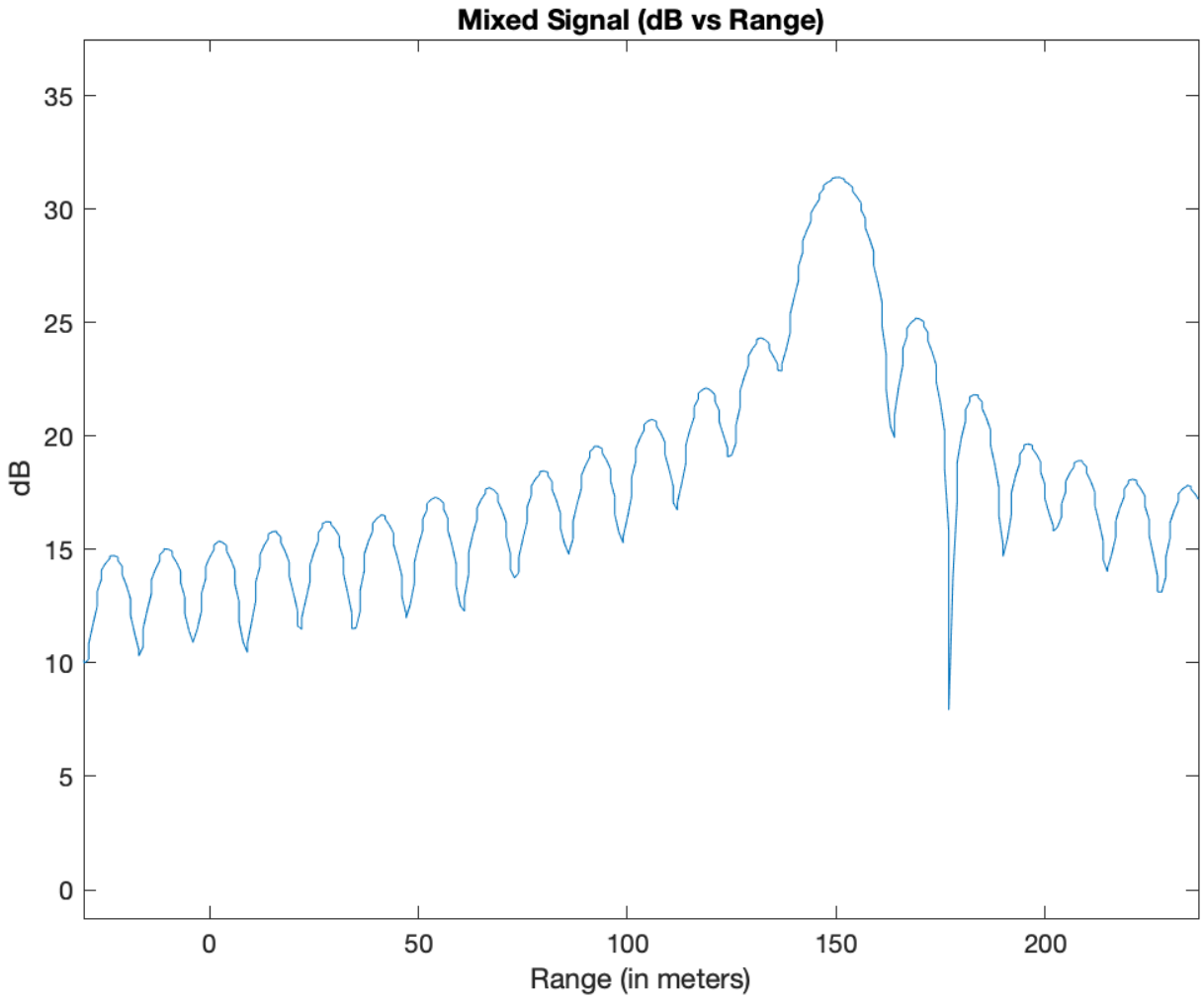

Figure 4.7: DFT of the mixed signal with the 150-meter coaxial cable, zoomed in.

where P_t is the total power supplied to the antenna and P_{rad} is the power that is radiated out into space. The remainder is dissipated as heat in the antenna [12].

Antenna efficiency is largely dependent on the impedance matching of the antenna to the source (the SDR in this case). Impedance matching is achieved when the source impedance is the complex conjugate of the load impedance. The easiest way to do this it to make the impedance completely resistive. However, this is impractical since it raises the radar noise figure, and so complex impedance circuits are used. If there is a mismatch between the load impedance (antenna) and the transmission line (SDR), part of the forward wave sent to the load is reflected along the

transmission line towards the source. This defines the reflection coefficient Γ

$$\Gamma = \frac{V_{\mathrm{r}}}{V_{\mathrm{f}}}, \tag{4.20}$$

where V_{f} is the complex amplitude of the forwarded wave and V_{r} is the complex amplitude of the reflected wave.

The mismatch of impedances results in standing waves along the transmission line which magnify transmission losses. The Voltage Standing Wave Ratio (VSWR) is a measurement of the depth of the standing waves, and therefore, a measurement of the impedance matching. VSWR is a function of the reflection coefficient Γ, which describes the power reflected from the antenna. The power reflected from the antenna represents the inefficiency of the antenna since the power is not being emitted. The VSWR is a metric that numerically describes how well the antenna's impedance matched to the radio or transmission line it is connected to. Thus, to increase the antenna efficiency, VSWR is minimized by matching the impedance of the antenna to the SDR. VSWR is given by

$$\mathrm{VSWR} = \frac{V_{\max}}{V_{\min}} = \frac{1+|\Gamma|}{1-|\Gamma|}, \tag{4.21}$$

where $V_{\max}$ and $V_{\min}$ are the resulting voltage maximum and minimum amplitude of the forward and reflective wave superimposed on each other.

To maximize the efficiency of the antenna, the Voltage Standing Wave Ratio (VSWR) is minimized by impedance matching the antenna with the SDR. The LimeSDR has an impedance of 50 ohms, which is the standard for transmitting radio equipment. To maximize the power delivered to the antenna and minimize the impedance mismatch between the antenna and free space of 377 ohms the antenna requires an impedance match of 50 ohms.

There are different methods for designing antennas to have a certain impedance. One technique is to use inductors or capacitors to change the impedance. This is sometimes done by introducing coils into a dipole antenna to increase the inductance. For a loop antenna, the shape of the antenna determines its impedance but also affects the directivity of the antenna (see Figure 4.8). The directivity of a full wave loop antenna is maximized when the loop antenna is a circle, however this shape results in an impedance of 133 ohms. A loop antenna shaped in a rectangle at

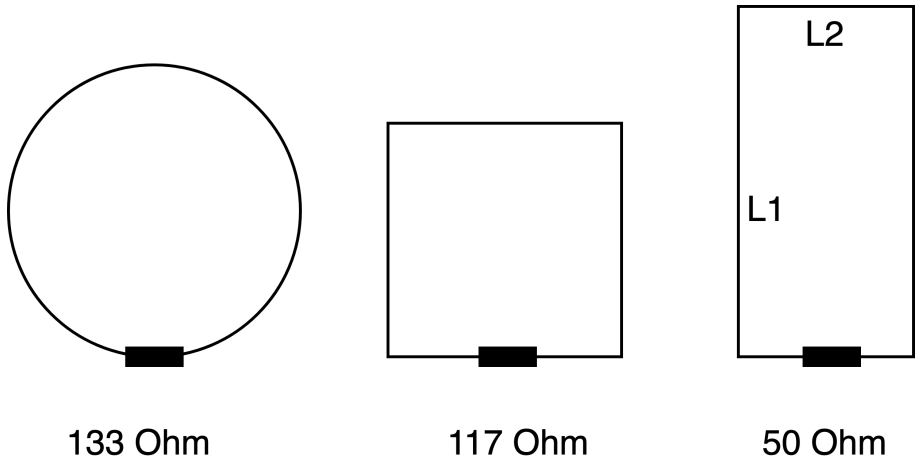

Figure 4.8: Different shaped antennas have different impedances, where $L1 = 2.028 \times L2$.

a length to width ratio of 2.028 has an impedance of 50 ohms. This matches with the impedance of the LimeSDR and minimizes the VSWR.

Resonant loop antennas (full loop antennas) are used for both the transmitter and the receiver in the LFMCW Radar and GPR design. The circumference of the full loop antenna is approximately one wavelength of the carrier frequency. This makes the antenna self-resonant at that frequency. The full wave loop can be thought of as two dipoles with the ends connected together. The exact length for resonance depends on the diameter of the conductor and the shape of the loop. For a ground-penetrating radar, the frequency usually is lower which results in a larger loop antenna. A longer wavelength can penetrate further through the ground. The transmit and receive loop antennas were made to be nearly identical and their resonant frequencies were measured using a spectrum analyzer. The SDR can then be configured to operate at the resonant frequency of the antennas.

Resonant loop antennas have a two-lobe radiation pattern and are most sensitive to radio waves broadside to the wires, with the nulls off the side (see Figure 4.9). The polarization of the signal is dependent on the position of the feedpoint on the loop. Feeding the signal through the bottom or the top of the loop antennas gives a horizontal polarization.

The resonant loop antennas are used since they are simple in design, provide a decently wide bandwidth, and are fairly efficient. This type of antennas isdi built using 6-gauge bare solid copper wire, with a coaxial cable. A coaxial cable is an electrical cable consisting of an inner

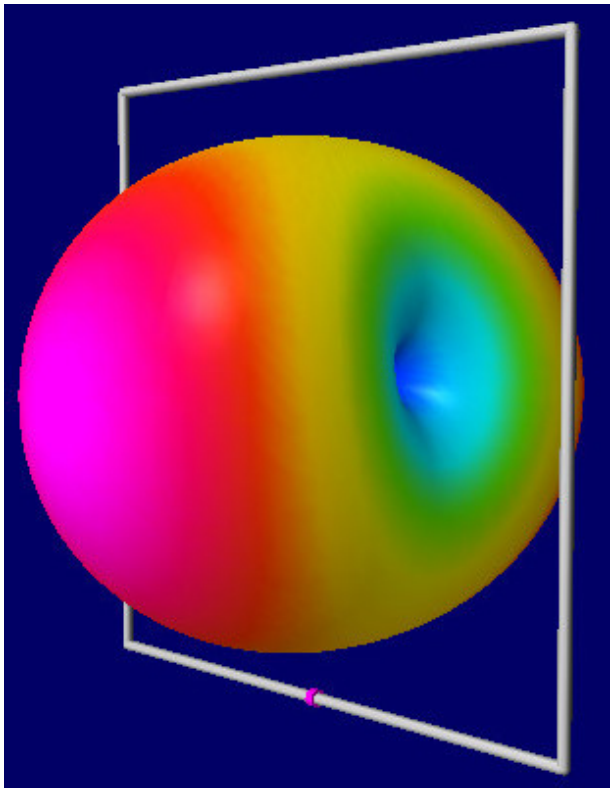

Figure 4.9: Radiation pattern of a resonant loop antenna. The loop is the white shape, with the feed at the pink point at the bottom. Peak gain is toward front or back of the loop

conductor surrounded by a concentric shield (braid). The word “coaxia” refers to the outer shield and the inner conductor sharing a geometric axis. The two conductors are separated by a dielectric (insulating material). There is also a protective outer sheath. To build the loop antenna, one end of the coaxial cable is an SMA connector that is used to connect to the LimeSDR-Mini. The other end is frayed and the shield is soldered to one endpoint of the loop antenna. The middle conductor is soldered to the other endpoint of the loop antenna (see Figure 4.10).

Both of the resonant loop antenna characteristics were tested using a spectrum analyzer (see Figure 4.11 and Figure 4.12). Antenna A has an approximate bandwidth of 40MHz. Antenna B has an approximate bandwidth of 42MHz. The efficient bandwidths of the antennas are the frequencies with a VSWR ratio less than 1.5. This is equivalent to the range of frequencies that results in less than four percent reflection power (see Table 4.3). Antenna A operates most efficiently at 700 MHz and Antenna B operates most efficiently at 704 MHz. Transmitting or receiving a chirp should be

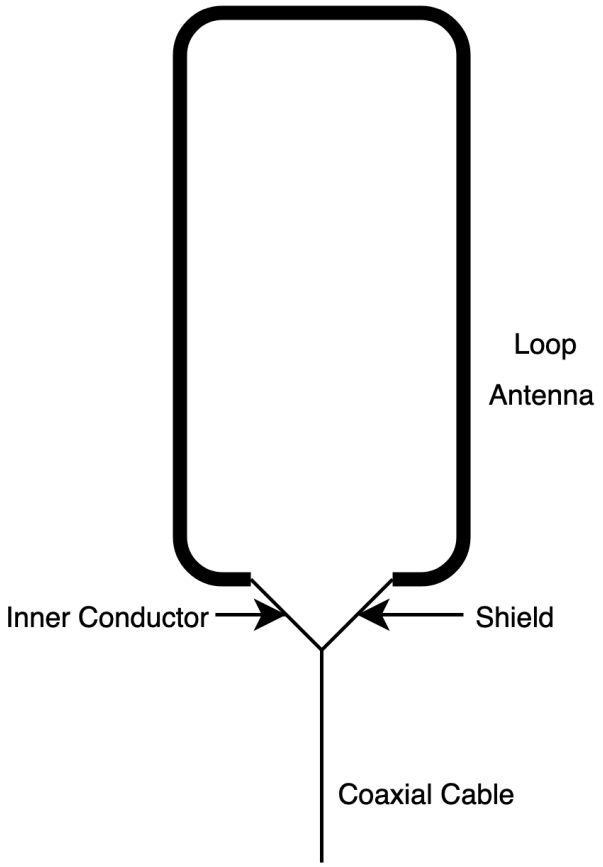

Figure 4.10: Diagram of a single loop antenna made from a 6-gauge cooper wire and coaxial cable.

at these frequencies. For example, a chirp with a bandwidth of 16MHz that begins at 694MHz and ends at 710MHz stays below a VSWR ratio of 1.5 on both antennas.

It is important to note that the resonant antenna frequency is dependent on the shape of the antenna. Antenna B when originally measured had a resonant frequency of 677MHz. Bending the loop antenna so that the endpoints are slightly closer changed the resonant frequency to 704MHz. Note that the antennas are sensitive to perturbations of their shape and need to be handled carefully.

4.6 Bleed-through Signal

Bleed-through signal (sometimes called the feed-through leakage) is a challenge in creating a working short-range radar. The bleed-through signal is the signal that leaks from the transmit-

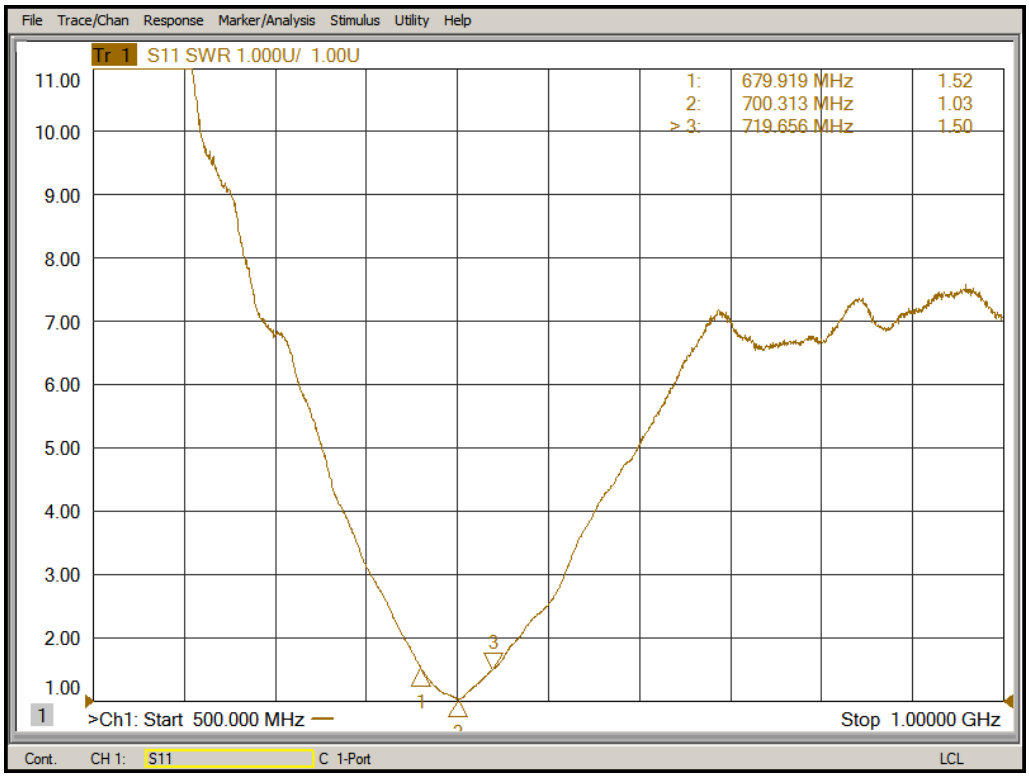

Figure 4.11: Antenna A: VSWR vs frequency from 500MHz to 1000MHz.

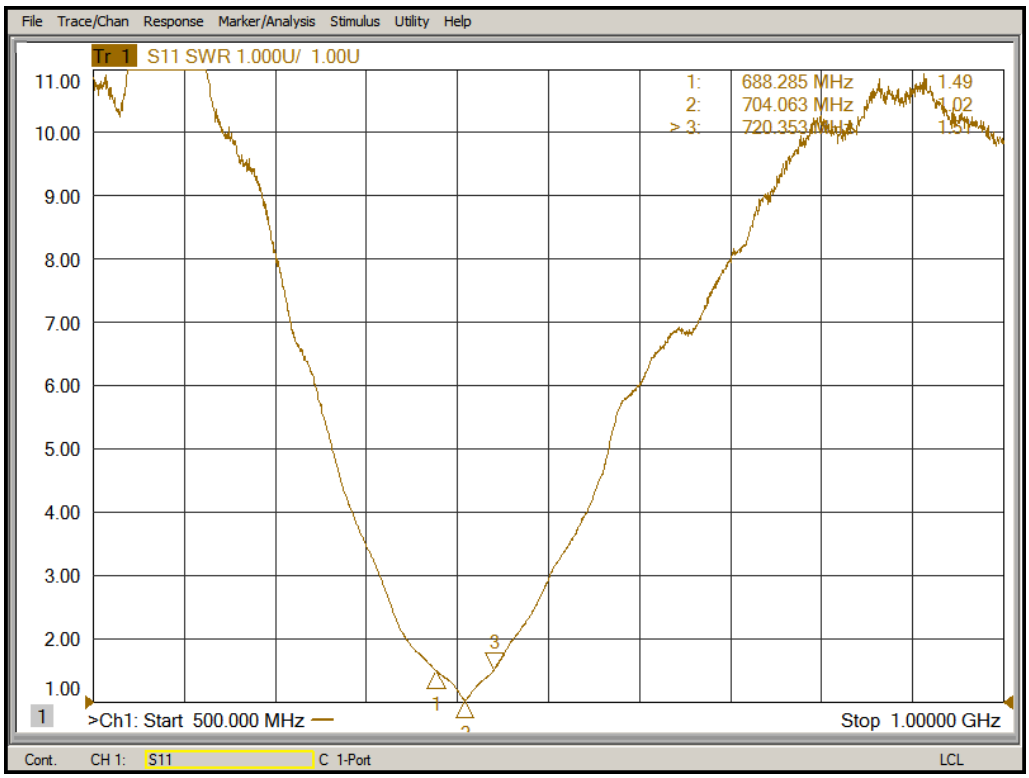

Figure 4.12: Antenna B: VSWR vs frequency from 500MHz to 1000MHz.

Table 4.3: Characteristics of the custom full loop antennas.

Antenna	Frequency (MHz)	VSWR (unitless)	Return Loss (dB)	Return Loss (%)
A	679.919	1.52	13.7	4.26
A	700.313	1.03	36.6	0.02
A	719.656	1.50	14.0	3.99
B	688.285	1.49	14.1	3.88
B	704.063	1.02	40.1	0.001
B	720.353	1.51	13.8	4.13

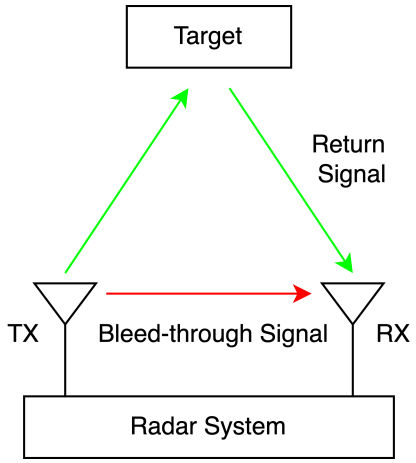

Figure 4.13: Diagram illustrating the difference between the bleed-through signal and the return signal. The receive signal is the summation of these two signals.

ter into the receiver (see Figure 4.13). The LFMCW radar system is constantly sending out and receiving electromagnetic waves. Some of the transmit signal leaks directly to the receiver either through the circuitry or through the antennas. Electronic circuits on the LimeSDR Mini have a limited dynamic range, and this leakage reduces the receiver sensitivity. The bleed-through signal impacts the ability to detect short-range targets. The received signal is the composition of the bleed-through signal and the return signal. The return signal from the target is buried under the bleed-through signal, resulting in targets not being detected. A simple test was conducted using antennas.

Since the bleed-through is the signal that leaks directly from the transmitter to the receiver, the bleed-through can be exploited to measure the FPGA time delay τ_{FPGA} and the resulting inter-

mediate frequency f_{FPGA}. Once f_{FPGA} is known, the bleed-through is no longer needed and should be removed from the receive signal so the return signal can be more accurately measured.

There are a couple of techniques used to address the bleed-through problem. Physically moving the antennas, changing the radar parameters, and signal processing on the data all impact the severity of the bleed-through signal.

One technique to minimize bleed-through is to move the antennas further apart. If the transmitter and receive are right next to each other, then they are essentially behaving as a 1:1 transformer. The physical size of the radar system limits how far the antennas can move apart. Since the GPR system will be mounted on an aerial quad-copter drone, the antennas cannot be placed very far apart.

When using aperture antennas, the transmitter can be placed offset from the receiver in range, so that the transmitter is in front of the receiver. This limits the amount of signal that bleeds through to the receiver. Unfortunately, full-wave loop antennas are bidirectional to the broadside of the loop wires (see Figure 4.9), so that offsetting the transmitter in front of the receiver has only a very limited effect on the bleed-through signal. However, this principle of minimizing the bleed-through signal by rearranging the antennas is still beneficial. By moving the receive antenna to be diagonally offset from the transmit antenna, the bleed-through signal is reduced.

I conducted a series of experiments using the different configurations show in Figure 4.14. The test was performed by only changing the antenna placement and all other variables were kept constant. The DFT of the mixed signal produces a peak at the frequency f_{FPGA}, which comes from the bleed-through signal. Going from configuration A to B (separating the antennas by 25 cm) resulted in the peak dropping 8.3 dB drop while going from configuration A to C resulted in a 10 dB drop.

Another technique to help prevent the target signal from being overpowered by the bleed-through signal is to taper the f_{FPGA} signal in short-range. This narrows the range of the 3 dB bandwidth of the signal peak at f_{FPGA}, allowing the f_{IF} signal of the target to appear with less power return. This is done by maximizing bandwidth as shown in Equation 4.13.

A high-pass filter can also be used to taper f_{FPGA} to reduce the bleed-through signal. A simple digital high pass filter in the time-domain can be expressed as $y[n] = x[n] - x[n-1]$ or as

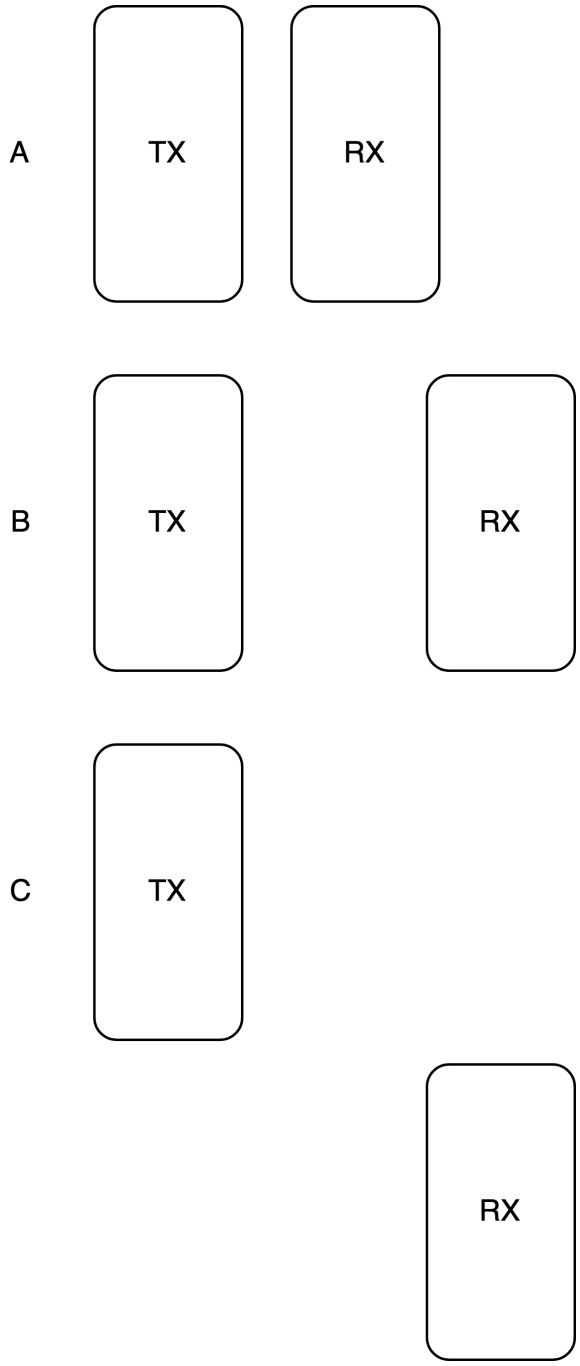

Figure 4.14: Antenna placement configurations: this figure shows the antennas when looking at the antennas from the target. The strength of the bleed-through signal is inversely related to the distance between the transmit and receive antennas.

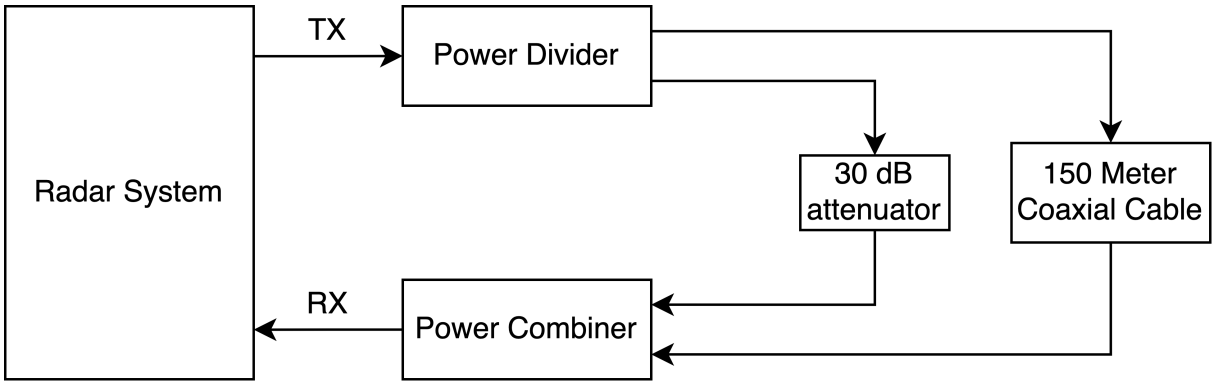

Figure 4.15: Setup of the feed-through nulling test with coaxial cables

$(1-z^{-1})X(z)$ in the Z-domain. This method attenuates the power bleed-through f_{FPGA} signal more than the target f_{IF} signal.

Although using these techniques lower the power return of the bleed-through signal, they do not adequately eliminate the bleed-through signal to reveal the return signal for very short-range. Digital signal processing techniques with more success are further explored in the next section.

4.7 Feed-through Nulling

Feed-through nulling is a signal processing technique used to eliminate the bleed-through signal from the receive signal. The received signal in the LFMCW radar is a combination of the bleed-through signal, the return echo from the target, and clutter return. The signal of interest is the return signal. Feed-through nulling is the process of subtracting the bleed-through signal so that the return echo can be discriminated.

To test the efficacy of feed-through nulling without complication of clutter, the coaxial cable setup shown in Figure 4.15 is used. The setup includes two lengths of coaxial cable that are attached to a power divider at the transmitter and a power combiner at the receiver. The short coaxial mimics the bleed-through signal and the 150-meter coaxial cable mimics the return signal. The long length of coaxial cable attenuates the signal significantly more than the short coaxial cable with a 30 dB attenuator, as it would be for when two antennas are used. This results in the receive signal being dominated by the signal passing through the short coaxial cable. All of the data produced in these tests were produced by using the parameters in Table 4.4. Figure 4.16

Table 4.4: Parameters used for the coaxial cable feed-through nulling test.

F_{carrier}	F_{s}	B	T_{c}
250MHz	36MHz	16MHz	100μs

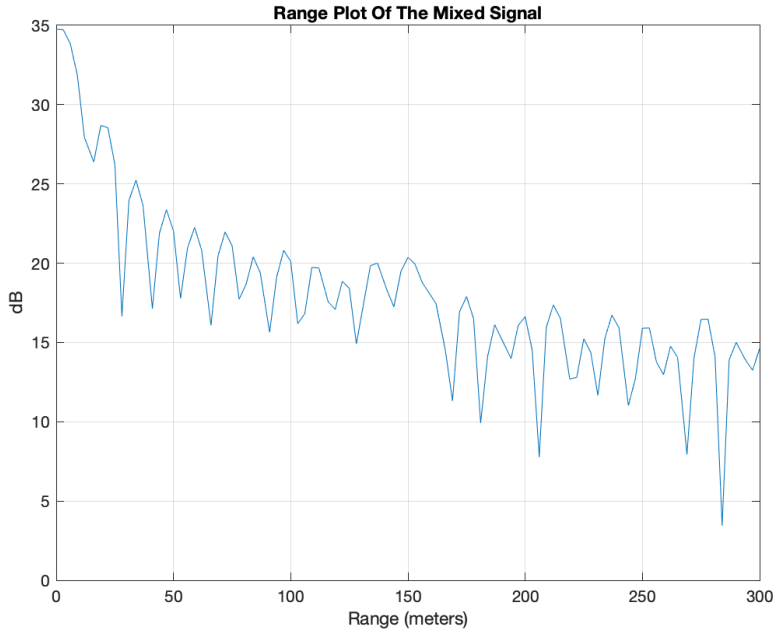

Figure 4.16: The resulting mixed signal is from mixing the RX signal with the TX signal. The x-axis is range instead of frequency. The f_{FPGA} corresponds to the zero point on the range scale. The 150-meter peak is obscured by the bleed-through signal.

shows the DFT of the mixed signal and shows the area of interest with the x-axis as the range. The intermediate frequency of f_{FPGA} is at 350 kHz. The figure shows the peak bleed-through at 35 dB with no evidence of a second peak corresponding to the target.

Three methods of feed-through nulling are identified and explored in the following.

1. **Synthesize and subtract out the frequency f_{FPGA} from the mixed signal.**

   The first feed-through nulling technique begins with the mixed signal, which is the receive signal that has been mixed with the transmit signal. The mixed signal produces two distinct frequencies: the intermediate frequency f_{FPGA} resulting from the bleed-through signal and the intermediate frequency f_{IF} resulting from the echo return from the target. These

frequencies are represented in the time-domain as $x_{\mathrm{B}}(t)$ and $x_{\mathrm{T}}(t)$ respectively, i.e.,

$$x_{\mathrm{B}}(t) = A_0 e^{j(2\pi f_{\mathrm{FPGA}} + \phi_0)} \tag{4.22}$$

$$x_{\mathrm{T}}(t) = A_1 e^{j(2\pi f_{\mathrm{IF}} + \phi_1)}. \tag{4.23}$$

The mixed signal $x_{\mathrm{M}}(t)$ is represented as

$$\begin{aligned} x_{\mathrm{M}}(t) &= x_{\mathrm{B}}(t) + x_{\mathrm{T}}(t) && (4.24) \\ x_{\mathrm{M}}(t) &= A_0 e^{j(2\pi f_{\mathrm{FPGA}} + \phi_0)} + A_1 e^{j(2\pi f_{\mathrm{IF}} + \phi_1)}, && (4.25) \end{aligned}$$

where A_0 and A_1 are the amplitudes of the frequency components.

This technique of feed-through nulling is to subtract out the bleed-through from the mixed signal so that the intermediate frequency f_{IF} dominates. To do this $x_{\mathrm{B}}(t)$ needs to be estimated. Three variables are estimated $x_{\mathrm{B}}(t)$: the f_{IF} frequency, phase offset ϕ_0 and the amplitude A_0.

The DFT of the mixed signal produces a peak at f_{IF} since the bleed-through signal is significantly stronger than the echo return signal. This gives the frequency f_{IF}. The $x_{\mathrm{B}}(t)$ is defined in Equation 4.22 with $A_0 = 1$ and $\phi_0 = 0$. The mixed signal and $x_{\mathrm{B}}(t)$ are shown in Figure 4.17.

To synthesize $x_{\mathrm{B}}(t)$ more accurately, the phase offset ϕ_0 is also calculated. The phase offset is needed because the LimeSDR-Mini is not coherent. The phase offset from mixing the transmit and receive chirp with the carrier frequency varies from run to run. Thus, the phase offset needs to be calculated for every time data is collected by running the software. The phase offset is found using the assumption that the bleed-through signal greatly overpowers the target signal ($A_0 \gg A_1$). The mixed signal $x_{\mathrm{M}}(t)$ is thus primarily made up of the bleed-through signal:

$$\begin{aligned} x_{\mathrm{M}}(t) &\approx x_{\mathrm{B}}(t) \\ x_{\mathrm{M}}(t) &\approx A_0 \cos(2\pi f_{\mathrm{FPGA}} t + \phi_0) + j A_0 \sin(2\pi f_{\mathrm{FPGA}} t + \phi_0). && (4.26) \end{aligned}$$

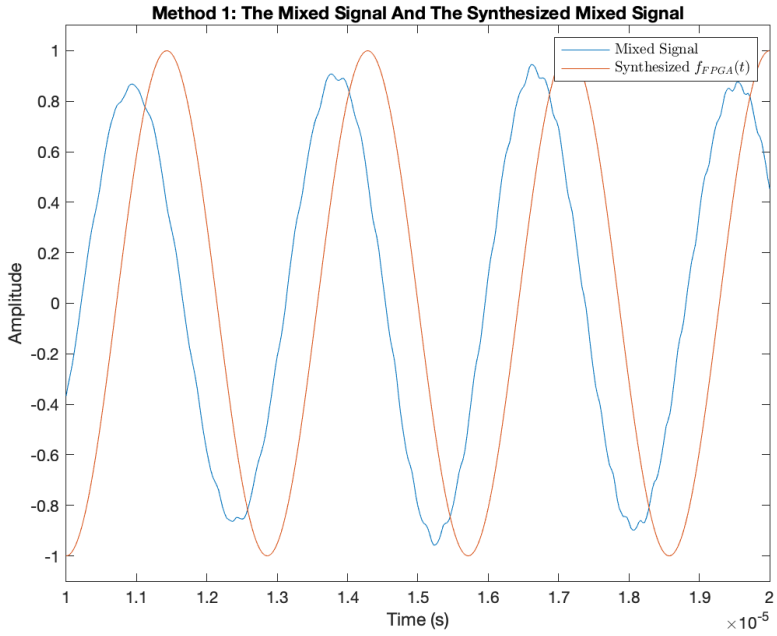

Figure 4.17: Zoomed in section of the real part of the mixed signal with the real synthesized $f_{\text{FPGA}}(t)$ function using the estimate f_{FPGA}. Compare with Figure 4.18.

From the assumption in Equation 4.26 the phase is isolated using the steps below,

$$\begin{aligned}
\frac{\text{imag}(x_{\text{M}}(t))}{\text{real}(x_{\text{M}}(t))} &= \frac{A_0 \sin(2\pi f_{\text{FPGA}} t + \phi_0)}{A_0 \cos(2\pi f_{\text{FPGA}} t + \phi_0)} \\
\frac{\text{imag}(x_{\text{M}}(t))}{\text{real}(x_{\text{M}}(t))} &= \tan(2\pi f_{\text{FPGA}} t + \phi_0) \\
\tan^{-1}\left(\frac{\text{imag}(x_{\text{M}}(t))}{\text{real}(x_{\text{M}}(t))}\right) &= 2\pi f_{\text{FPGA}} t + \phi_0 \\
\phi_0 &= \tan^{-1}\left(\frac{\text{imag}(x_{\text{M}}(t))}{\text{real}(x_{\text{M}}(t))}\right) - 2\pi f_{\text{FPGA}} t. \qquad (4.27)
\end{aligned}$$

This provides the phase difference for every data sample. The mean of the phase differences is used for ϕ_0. The $x_{\text{B}}(t)$ can now be defined in Equation 4.22 with $A_0 = 1$. The mixed signal and $x_{\text{B}}(t)$ with the updated phase shift is shown in Figure 4.18

The amplitude of the signal varies because of the VSWR characteristics of the antennas, which vary based on the frequency. This results in varying amplitude of the chirps and results in a mixed signal with varying amplitudes of the intermediate frequencies (see Figure 4.19). The amplitude is thus a function of time so that Equation 4.22 is more accurately

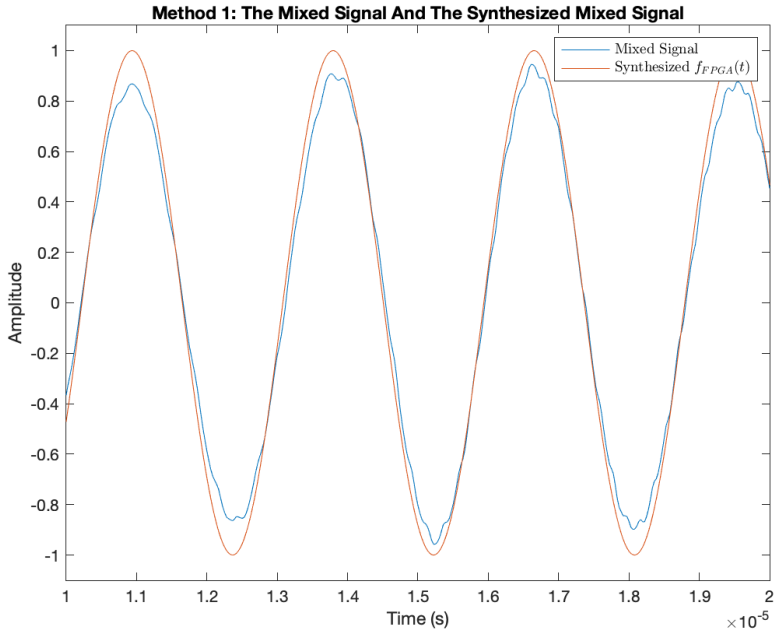

Figure 4.18: Zoomed in section of the real part of the mixed signal with the synthesized $x_{\mathrm{B}}(t)$ function using the estimates f_{FPGA} and ϕ_0. Compare with Figure 4.17

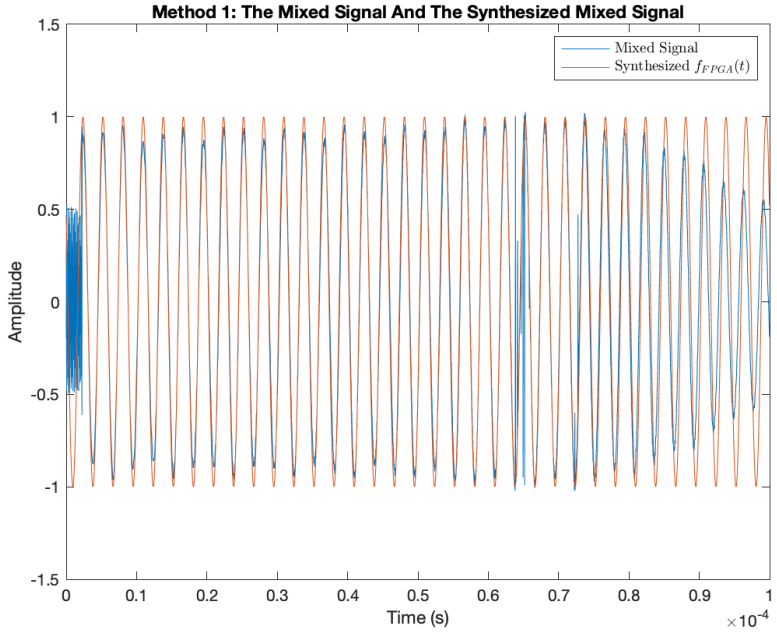

Figure 4.19: The real part of the mixed signal with the synthesized $x_{\mathrm{B}}(t)$ function using the estimates f_{FPGA} and ϕ_0. The amplitude of the mixed signal varies over time.

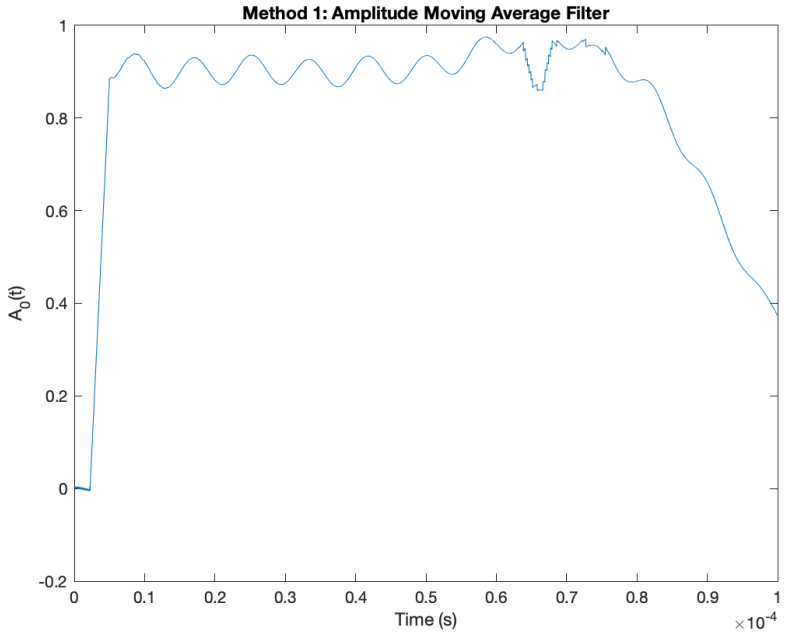

Figure 4.20: The moving average filtered ratio of the mixed signal and $x_{\mathrm{B}}(t)$ function using the estimates f_{FPGA} and ϕ_0.

represented as

$$x_{\mathrm{B}}(t) = A_0(t) \exp j(2\pi f_{\mathrm{FPGA}} + \phi_0), \tag{4.28}$$

where $A_0(t)$ is the signal amplitude function. $A_0(t)$ is estimated using a simple moving average (boxcar) filter of the ratio of the mixed signal and the $f_{\mathrm{FPGA}}(t)$ function with an amplitude of one, and a window sample size of 100 samples

$$A_0(t) \approx \mathrm{MovingAverageFilter}\left(\frac{x_{\mathrm{M}}(t)}{\exp j(2\pi f_{\mathrm{FPGA}} + \phi_0)}\right). \tag{4.29}$$

The moving average filter output is plotted in Figure 4.20.

The time-domain function $f_{\mathrm{FPGA}}(t)$ is recreated with the frequency found from the peak in the DFT of the mixed signal, the phase found in Equation 4.27, and the amplitude found in Equation 4.29.

The estimated $x_B(t)$ is subtracted from the mixed signal $x_M(t)$ to produce a new mixed signal with a lower bleed-through signal power (see Figure 4.22). This reveals the $x_{\mathrm{T}}(t)$ signal from the 150-meter length of the coaxial cable. The mixed signal with feed-through nulling

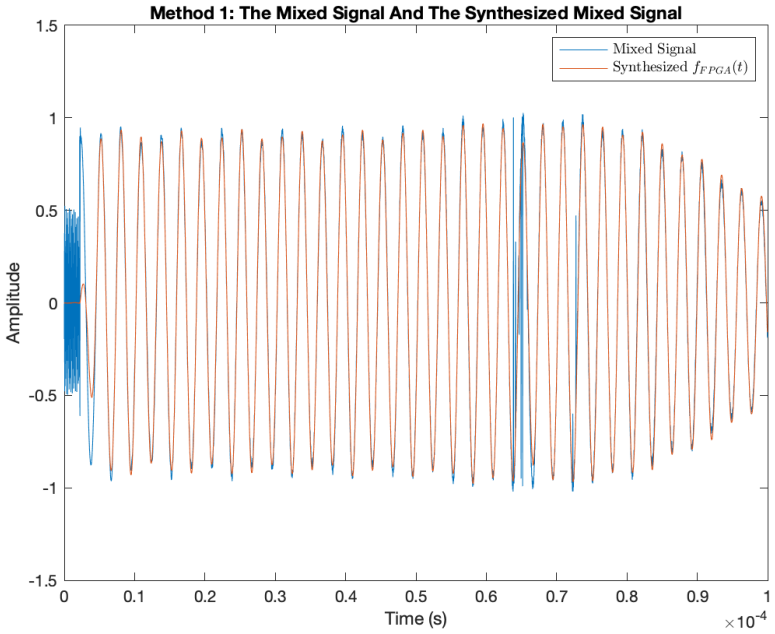

Figure 4.21: The real part of the mixed signal with the synthesized $x_{\mathrm{B}}(t)$ function using the estimates f_{FPGA}, ϕ_0, and $A_0(t)$.

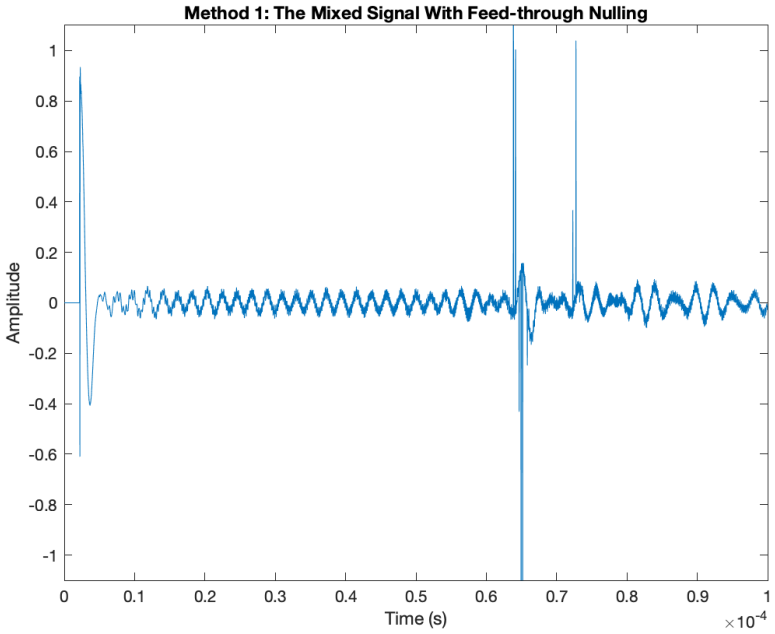

Figure 4.22: The real part of the mixed signal with feed-through nulling applied.

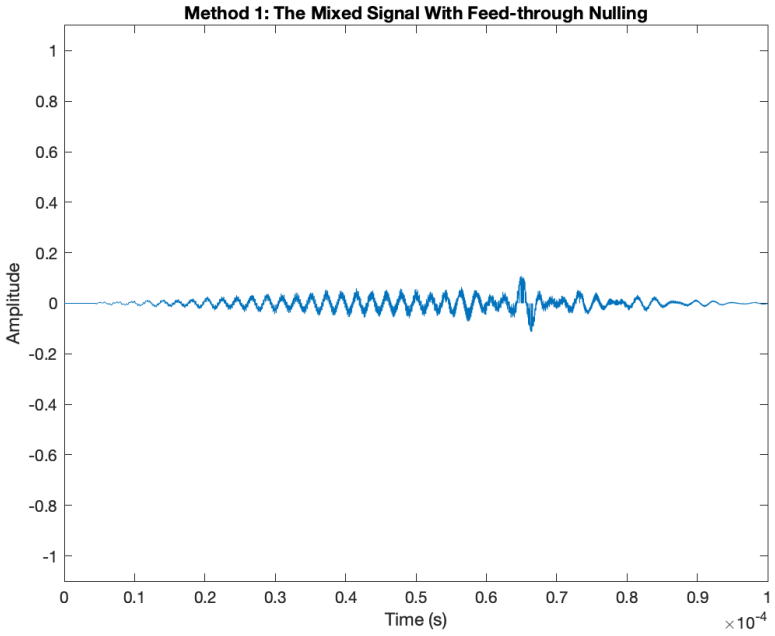

Figure 4.23: The real part of the mixed signal with feed-through nulling, zeroed data points, and the Hamming window applied.

applied has seemingly random peaks. To improve upon the signal, the data with abnormal peaks are zeroed out. This eliminates spectral leakage and makes for a cleaner signal. In addition, a Hamming window is applied to the signal to smooth it out. This results in Figure 4.23. The DFT of the mixed signal with feed-through nulling applied (Figure 4.24) reveals the length of the 150-meter coaxial cable. The intermediate signal f_{FPGA} corresponds with 0 meters. This feed-through nulling technique results in the peak of the mixed bleed-through signal dropping by 23.5 dB and revealing the range of the coaxial cable at 17.9 dB.

2. **Synthesize and subtract out the bleed-through signal chirp from the receive signal.** The second technique of feed-through nulling is to synthesize and subtract out the bleed-through signal from the receive signal (before mixing). This computation is similar to the previous technique. The bleed-through signal is synthesized by recreating the transmit chirp and then using transforms so that it matches the bleed-through signal. The first step is to circularly shift the samples of the chirp to match the τ_{FPGA} time delay. This is done by taking the DFT of the mixed signal and converting the DFT peak f_{FPGA} to τ_{FPGA} with Equation 4.9:

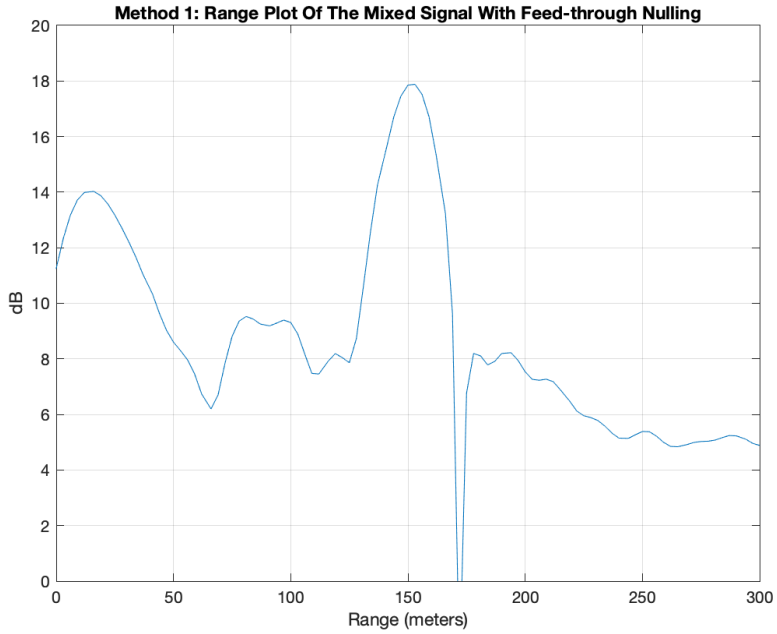

Figure 4.24: Feed-through nulling method 1: The DFT of the mixed signal with feed-through nulling and other signal processing applied.

$\tau_{\text{FPGA}} = \frac{f_{\text{FPGA}}}{S}$. The number of samples n to circularly shift is the same number of samples that occur in the time delay τ_{FPGA}: $n = \text{Floor}(\tau_{\text{FPGA}}\, F_s)$.

Since samples are taken at discrete time steps, samples can only be shifted in integers. This means the synthesized bleed-through signal phase still needs to be corrected. The phase shift ϕ_0 is found by taking the receive signal and dividing it by the current synthesized bleed-through signal. This gives a phase difference at all samples, so the mean of the differences is taken.

The amplitude of the synthesized bleed-through signal is adjusted by using an equation similar in principle to Equation 4.29. However, the receive signal is used instead of the mixed signal. If the receive signal is represented as $x_{\text{R}}(t)$, the amplitude is

$$A_0(t) \approx \text{MovingAverageFilter}\left(\frac{x_{\text{R}}(t)}{\exp\left(j\left[2\pi\left\{\frac{S}{2}\,t^2 + f_0\,t\right\} + \phi_0\right]\right)}\right). \tag{4.30}$$

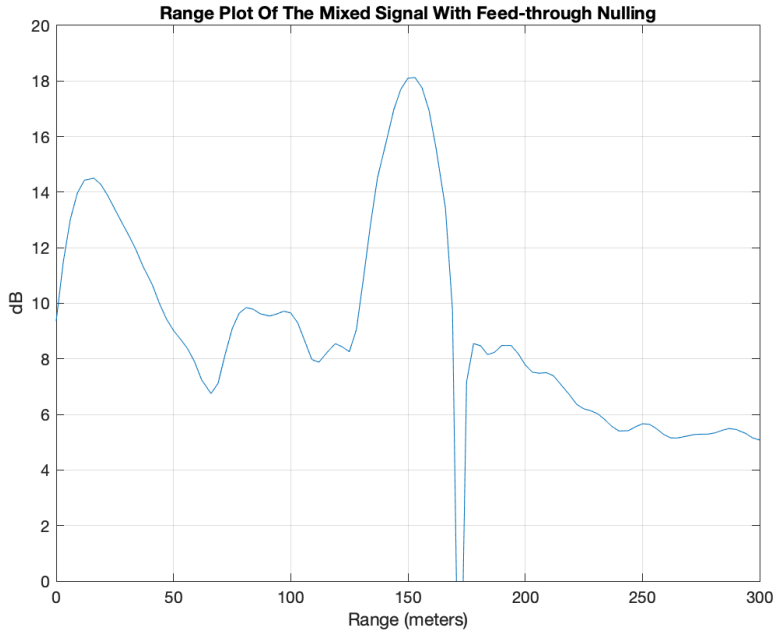

Figure 4.25: Feed-through nulling method 2: The DFT of the mixed signal with feed-through nulling and other signal processing applied.

The synthesized bleed-through signal is

$$x_{\mathrm{B}}(t) = A_0(t)\exp\left(j\left[2\pi\left\{\frac{S}{2}\,t^2 + f_0\,t\right\} + \phi_0\right]\right). \tag{4.31}$$

Subtracting the synthesized bleed-through signal from the receive signal results in a signal that is dominated by the signal coming from the 150-meter coaxial cable. Further signal processing is applied to the signal, identical to the feed-through nulling technique with the mixed signal. Then, random signal spikes zeroed out to eliminate spectral leakage and a Hamming filter is applied to smooth out the signal. The results are shown in Figure 4.25. The peak of the DFT mixed bleed-through signal drops 25.6 dB and the signal from the 150-meter results in a peak of 18.1 dB. This is a slight improvement from the previous feed-through nulling method.

3. **Measure and subtract out the bleed-through signal chirp from the receive signal.** The last feed-through nulling technique is to measure the bleed-through signal and subtract the bleed-through signal from the received signal with the target. A simple way to do this is

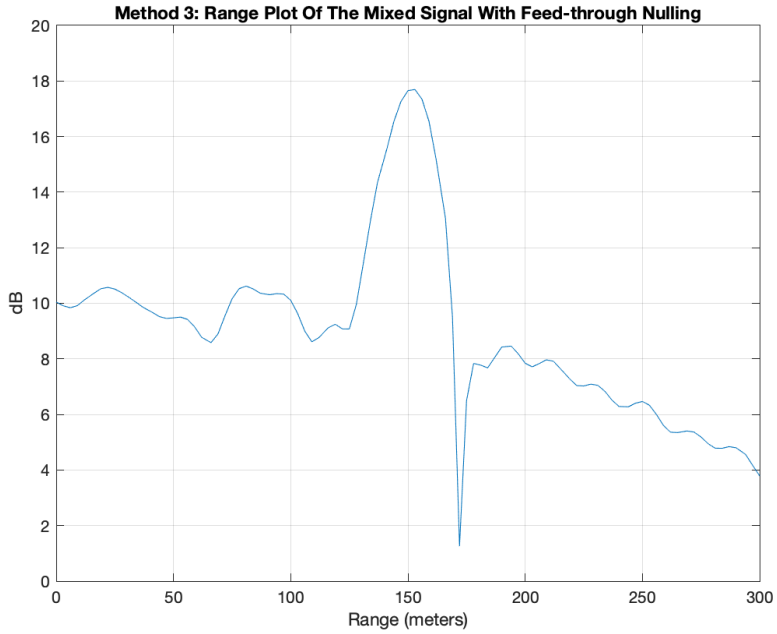

Figure 4.26: Feed-through nulling method 3: The DFT of the mixed signal with feed-through nulling and other signal processing applied.

to capture a signal with no target present, and then capture a signal with the target present. Subtracting one signal from the other theoretically should result in a signal with the target signal without a bleed-through signal. This technique has the greatest potential to reveal the target signal. It not only eliminates the bleed-through signal but also the clutter (objects that are not the target). However, this method does not work perfectly in practice because the LimeSDR-Mini carrier frequency is not coherent, so the signal varies slightly from run to run. In addition, the environment slightly changes from run to run. Thus, this method is only useful in testing. In the coaxial cable test set up, this method is done by capturing the signal only using the short connected coaxial cable, and then subtracting that signal from the receive signal when both cables are connected as shown in Figure 4.15. The results of this feed-through technique are shown in Figure 4.26. The power of the bleed-through signal decreased by 24.7 dB and the signal from the 150-meter cable results in a peak of 17.7 dB.

Since the LimeSDR-mini varies from run to run, this technique has limited effectiveness. Multiple instances of the bleed-through signal are captured (with the one short coaxial cable) and multiple instances of the receive signal (with both the short and long coaxial cable)

are captured. If both signals captured perform similarly, then this feed-through nulling works well and does the best job at eliminating the bleed-through signal. It also has the advantage of eliminating signal returns from clutter, while the other feed-through techniques only eliminate the bleed-through signal.

In my tests, the feed-through nulling techniques reduce the bleed-through signal enough to reveal the signal coming from the 150-meter coaxial cable. These techniques can be applied when detecting a target with antennas.

CHAPTER 5. RESULTS

5.1 LFMCW Radar Target Range Results

With the LimeSDR Mini radar system verified and able to accurately measure the length of the 150-meter coaxial cable, the radar system is outfitted with loop antennas to measure the range of a target (see Figure 5.1). A custom corner reflector is used as a target to maximize the power of the return signal. A corner reflector consists of three perpendicular sides. The three sides have a square shape and are covered with a layer of aluminum foil to reflect better. Due to the shape and material of the corner reflector, electromagnetic waves are reflected directly back towards the source. In addition to the corner reflector, a 15-passenger van was also parked directly behind the corner reflector to provide a larger radar cross-section further away.

A selected set of data results show success in detecting the target. The feed-through technique that worked the best was when the receive signal is captured without the target and then with the target. The feed-through nulling technique subtracts out the receive signal without the target from the receive signal with the target. This not only reduces the bleed-through signal but also helps eliminate clutter. The radar was tested at multiple ranges and correctly detected the target (See Figure 5.2, 5.3, and 5.4). The largest peak of the DFT of the mixed signal corresponds to the target. The data was collected with the radar parameters: $F_s = 36$ MHz, $B = 16$ MHz, and $T_c = 100$ μs.

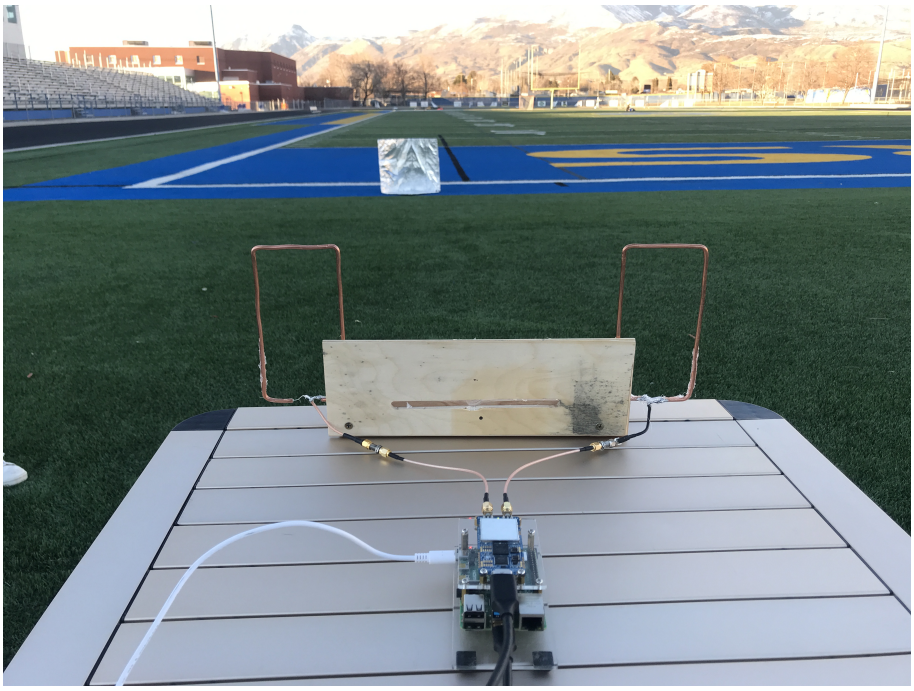

Figure 5.1: LFMCW Radar Range Testing. The distance between the radar and the target is 17m-34m.

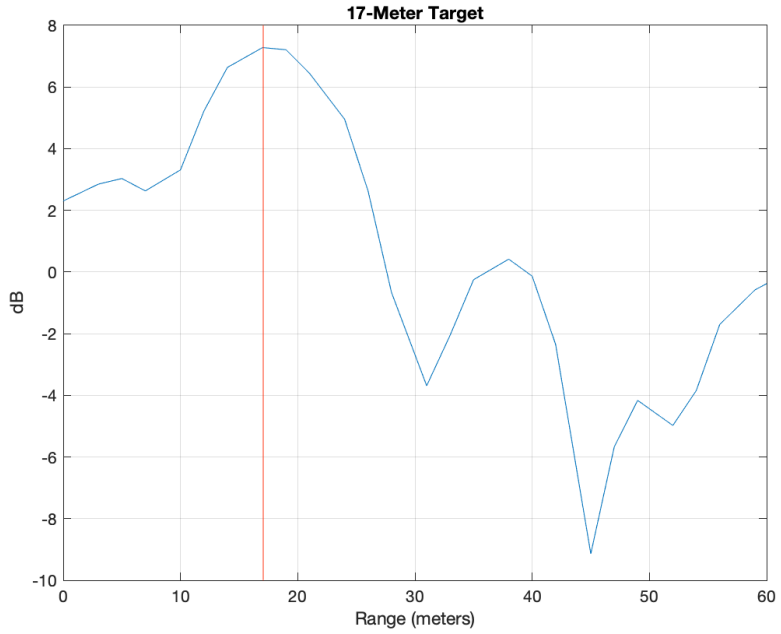

Figure 5.2: The LFMCW radar range test results at 17 meters.

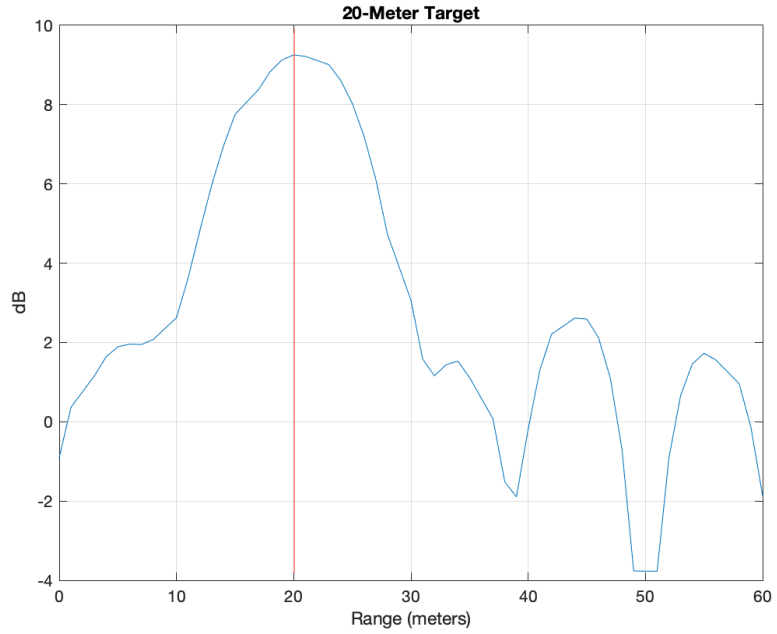

Figure 5.3: The LFMCW radar range test results at 20 meters.

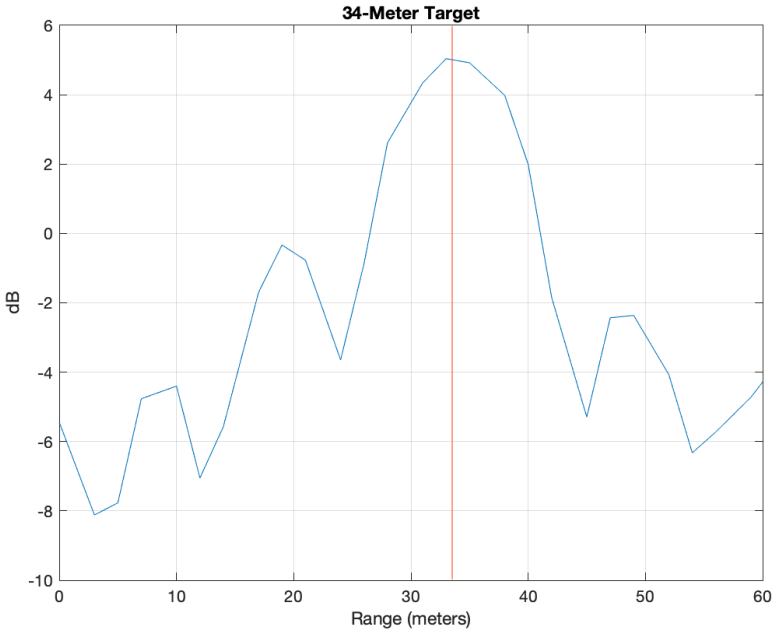

Figure 5.4: The LFMCW radar range test results at 34 meters.

CHAPTER 6. CONCLUSION

6.1 Conclusion

Short-range targets were successfully detected by the prototype LFMCW radar. The radar was implemented using the LimeSDR-Mini, the single-board Raspberry Pi 4 computer, and custom loop antennas. The goal of this paper was to see if the LimeSDR board could adequately be used for a ground penetrating radar system. Although there were instances where the LFMCW radar worked well, it did not work reliably since the signal varies significantly from run to run. The LimeSDR board works well for digital communication, but the hardware is limited enough to rule out its use as a reliable working short-range radar. The key problem with the LimeSDR is its instability. This results in synchronization issues of the transmitter and receiver, differences in signals from run to run, and bad data being randomly generated.

The problem with the lack of timing synchronization of the transmitter and receiver in the hardware can be mitigated in signal processing to some degree, as long as the delay is known. Once the executable on the software is running, the delay is consistent. This delay can be measured in real-time and used to overcome the lack of synchronization. However, the instability of the LimeSDR occasionally results in a time delay so large that the data is unusable.

The LimeSDR transmits and receives slightly different signals every time the same executable runs. About 25% of the time the data is corrupted. In addition, the LimeSDR is not coherent so the phase of the signal will be different on every run. This makes capturing the clutter without the target also inconsistent. Due to the fact this is a short-range radar, feed-through nulling is necessary to detect the target. However, the variability of the LimeSDR makes feed-through nulling unreliable. Data sets generated between different runs of software cannot be compared, as the signals used are not identical.

To develop a working LFMCW radar, the chirp bandwidth needs to be large enough to have a fairly accurate range bin. A large bandwidth requires a large sampling rate and the LimeSDR performs more inconsistently as the sampling rate increases.

All of these issues are exacerbated in a short-range radar. When a target is far enough from the LFMCW radar, the intermediate frequency produced clears any interference from the bleed-through signal. However, for a short-range radar, the target echo is buried in the bleed-through signal. Additionally, slight variances in the FPGA time delay in a long-range radar have less of an impact on the results.

These issues, while some can be mitigated, may prevent the LimeSDR boards from being the basis of a reliable radar. However, as a proof of concept, the potential of using a low-cost SDR for a radar system was a success.

6.2 Future Work

The requirements of the antennas are to be compact, operate at a low frequency, and operate at a large bandwidth. The loop antennas are able to meet these criteria, but the bleed-through signal became a significant issue. Further research and development in other types of antennas could reduce the severity of the bleed-through signal.

While the LimeSDR board series does not have sufficient capabilities to be used to support a reliable radar, other brands of SDR can be explored. The USRP SDR should be used since it is greater in capabilities. The USRP devices have been around for longer and there is significantly more research on radar systems built using USRP SDRs.

The Raspberry Pi image used to configure the LimeSDR also has support for USRP devices. Since the LimeSDR-Mini was configured using the LimeSuite Library, the radar code cannot be compiled to be used on the USRP SDR. However, the USRP has the UHD C++ library which is very similar to the LimeSuite Library. The majority of the code can be refactored. Further development on the USRP B10 SDR board is needed to show that an SDR is capable of performing as a reliable radar.

REFERENCES

[1] J. Marimuthu, K. S. Bialkowski, and A. M. Abbosh, "Software-Defined Radar for Medical Imaging," *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 2, pp. 643–652, Feb. 2016. 2

[2] J. Meier, R. Kelley, B. M. Isom, M. Yeary, and R. D. Palmer, "Leveraging Software-Defined Radio Techniques in Multichannel Digital Weather Radar Receiver Design," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 6, pp. 1571–1582, Jun. 2012. 2

[3] H. Zhang, L. Li, and K. Wu, "24GHz Software-Defined Radar System for Automotive Applications," in *2007 European Conference on Wireless Technologies*, Oct. 2007, pp. 138–141. 2

[4] A. Lestari, D. D. Patriadi, I. H. Putri, B. Harnawan, O. D. Winarko, W. Sediono, and M. A. K. Titasari, "FPGA-based SDR implementation for FMCW maritime surveillance radar," in *2017 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, Oct. 2017, pp. 15–20. 2

[5] M. Pérez Cerquera, J. Colorado, and I. Mondragón, "UAV for Landmine Detection Using SDR-Based GPR Technology," in *Robots Operating in Hazardous Environments*. IntechOpen, 2017. 2

[6] J. Park, J. T. Johnson, N. Majurec, M. Frankford, E. Culpepper, J. Reynolds, J. Tenbarge, and L. Westbrook, "Software defined radar studies of human motion signatures," in *2012 IEEE Radar Conference*, May 2012, pp. 0596–0601, iSSN: 2375-5318. 2

[7] C. W. Rossler, E. Ertin, and R. L. Moses, "A software defined radar system for joint communication and sensing," in *2011 IEEE RadarCon (RADAR)*, May 2011, pp. 1050–1055, iSSN: 2375-5318. 2

[8] K. Stasiak and P. Samczynski, "FMCW radar implemented in SDR architecture using a USRP device," in *2017 Signal Processing Symposium (SPSympo)*, Sep. 2017, pp. 1–5. 2, 4

[9] T. P. Wibowo and F. Y. Zulkifli, "Design of FMCW Ground Penetrating Radar For Concrete Inspection At ISM Band 2.4–2.5 GHz," in *2019 IEEE Asia-Pacific Microwave Conference (APMC)*, Dec. 2019, pp. 1232–1234. 4

[10] S. Costanzo, F. Spadafora, A. Borgia, H. O. Moreno, A. Costanzo, and G. Di Massa, "High Resolution Software Defined Radar System for Target Detection," Oct. 2013, iSSN: 2090-0147 Pages: e573217 Publisher: Hindawi Volume: 2013. [Online]. Available: https://www.hindawi.com/journals/jece/2013/573217/ 4

[11] H. Zhang, L. Li, and K. Wu, "Software-Defined Six-Port Radar Technique for Precision Range Measurements," *IEEE Sensors Journal*, vol. 8, no. 10, pp. 1745–1751, Oct. 2008. 4

[12] F. Ulaby and D. Long, *Microwave Radar and Radiometric Remote Sensing*. The University of Michigan Press, 2014. 10, 28

[13] D. M. Schroeder, R. G. Bingham, D. D. Blankenship, K. Christianson, O. Eisen, G. E. Flowers, N. B. Karlsson, M. R. Koutnik, J. D. Paden, and M. J. Siegert, "Five decades of radioglaciology," *Annals of Glaciology*, vol. 61, no. 81, pp. 1–13, 2020, edition: 2020/03/09 Publisher: Cambridge University Press. [Online]. Available: https://www.cambridge.org/core/article/five-decades-of-radioglaciology/BB862CCC314A993C14B3C6F62BD4F0B6 10

[14] J. A. Dowdeswell and S. Evans, "Investigations of the form and flow of ice sheets and glaciers using radio-echo sounding," *Reports on Progress in Physics*, vol. 67, no. 10, pp. 1821–1861, Aug. 2004, publisher: IOP Publishing. [Online]. Available: https://doi.org/10.1088/0034-4885/67/10/r03 10

[15] J. L. Bamber, J. A. Griggs, R. T. W. L. Hurkmans, J. A. Dowdeswell, S. P. Gogineni, I. Howat, J. Mouginot, J. Paden, S. Palmer, E. Rignot, and D. Steinhage, "A new bed elevation dataset for Greenland," *The Cryosphere*, vol. 7, no. 2, pp. 499–510, 2013. [Online]. Available: https://tc.copernicus.org/articles/7/499/2013/ 11

[16] R. Easton, *Fourier Methods in Imaging*, ser. The Wiley-IS&T Series in Imaging Science and Technology. Wiley, 2010. [Online]. Available: https://books.google.com/books?id=QuIHjnXQqM8C 14

[17] V. Tuzlukov, *Signal Processing Noise*, ser. Electrical Engineering & Applied Signal Processing Series. CRC Press, 2018. 19

[18] M. Richards, J. Scheer, and W. Holm, *Principles of modern radar: Basic principles*. SciTech Publishing, 2010. 19

APPENDIX A. SOFTWARE LIBRARIES

A.1 Lime Suite Overview

This appendix gives more resources and explains in more detail how the code interacts with the hardware. All resources mentioned in this appendix can be found in Table A.3.

The software used to configure the LimeSDR is the Lime Suite library. Lime Suite is a collection of software supporting several hardware platforms including the LimeSDR, drivers for the LMS7002M transceiver RFIC, and other tools for developing with LMS7-based hardware. Lime Suite has a C++ library (in addition to a GUI) to configure the hardware and has a well-documented API (Application Programming Interface).

A good start to understanding the Lime Suite (LMS) API is to read through the quick start guide. This guide does an excellent job at explaining how to set up the device and to do simple transmits and receives. Example code is also provided in the Lime Suite GitHub repository.

A.2 Configurable LimeSDR Parameters

- Sampling frequency: f_s:
- Oversample Scaler: The hardware samples at OVERSAMPLE_SCALER $\cdot f_s$ and then decimates the samples back to f_s,
- Reference clock
- throughputVsLatency: bias towards low latency or high throughput
- Individual Channel Parameters
  - RX Antenna: can be manually set or automatically set.
    - LNAH: 2 GHz $\leq f \leq$ 3.5 GHz

* LNAW: 10 MHz $\leq f \leq$ 2 GHz

‘

- TX Antenna: can be manually set or automatically set.
  * Band1 2 GHz $\leq f \leq$ 3.5 GHz
  * Band2 10 MHz $\leq f \leq$ 2 GHz
- Gain: set with normalized gain [0,1] or dB gain [0,73]
- FIFO buffer size
- Read N Samples from Receive FIFO (removes samples from FIFO)
- Write N Samples to Transmit FIFO
- Set timestamp in Transmit FIFO (removes samples from FIFO when timestamp occurs)

A.3 Data Types and Stream Protocol

A sample is a real value and a complex value (sample = I + Q). The I and Q samples are represented in a buffer interleaved: IQIQIQ.... Depending on the the data type a sample can be 24 bit, 32 bits, or 64 bits.

The digital-to-analog converter (DAC) and analog-to-digital converter (ADC) have 12-bit resolution. This means that there are only 4096 values that can be represented. Data types represent these values differently.

There are three data types that can be used to represent the samples. The LMS_FMT_I12, LMS_FMT_I16, and the LMS_FMT_F32 data type. The LMS_FMT_I12 is a data type that best represents the DAC. A sample is a 12-bit real value and a 12-bit complex value that is in the range [-2048,2047]. The hardware does not support a 12-bit data structure, so it is stored in 16-bit integer values. The advantage of using the LMS_FMT_I12 data type is apparent in the USB data transfer to and from the LimeSDR and Raspberry Pi computer. A sample (I(12-bit) + Q(12-bit)) is transferred using three bytes. The packet layout for this type can be found in Table A.1. There is no link throughput waste, but there is more processing involved on the processing side for I and Q samples unscrambling. Choosing this compressed format can decrease link use at the expense of additional processing on the Raspberry Pi.

Table A.1: Stream protocol payload format for 12-bit compressed samples

Byte Index	Bits	Description
0	7-0	ch0_I0 [7:0]
1	3-0	ch0_I0 [11:8]
	7-4	ch0_Q0 [3:0]
2	7-0	ch0_Q0 [11:4]
3	7-0	ch1_I0 [7:0]
4	3-0	ch1_I0 [11:8]
	7-4	ch1_Q0 [3:0]
5	7-0	ch1_Q0 [11:4]
6	7-0	ch0_I1 [7:0]
7	3-0	ch0_I1 [11:8]
	7-4	ch0_Q1 [3:0]
8	7-0	ch0_Q1 [11:4]
9	7-0	ch1_I1 [7:0]
10	3-0	ch1_I1 [11:8]
	7-4	ch1_Q1 [3:0]
...	...	...

The LMS_FMT_I16 data type is the LMS_FMT_I12, but is shifted over left four bits, so that the four least significant bits are zeros. A sample in the LMS_FMT_I16 is a 16-bit real value and a 16-bit complex value that is in the range [-32768,32767]. In the Stream Protocol packet for the USB data transfer, the data is sent in a sample (I(16-bit) + Q(16-bit)) of four bytes. This is more inefficient than the LMS_FMT_I12 data type since 4 bits in the real part and 4 bits in the imaginary part of the sample are not used. The packet layout for this type can be found in Table A.2. The advantage of using the LMS_FMT_I16 is that even if there is throughput waste, there is no I and Q unscrambling on the processing side.

The LMS_FMT_F32 data type is the same as the LMS_FMT_I16 data type inside the LimeSDR hardware and in the stream protocol. However, on the software side of the Raspberry Pi the LMS_FMT_I16 is normalized to the LMS_FMT_F32 to float values of [-1:1]. This is done internally by dividing by LMS_FMT_I16 by 0x7FFF. This data type is useful when performing signal processing, but requires more processing and takes up more memory. When saving the data from the LimeSDR into a file on the Raspberry Pi, the LMS_FMT_I12 or LMS_FMT_I16 data type

Table A.2: stream protocol payload format for 16-bit compressed samples

Byte Index	Bits	Description
0	7-0	ch0_I0 [7:0]
1	7-0	ch0_I0 [15:8]
2	7-0	ch0_Q0 [7:0]
3	7-0	ch0_Q0 [15:8]
4	7-0	ch1_I0 [7:0]
5	7-0	ch1_I0 [15:8]
6	7-0	ch1_Q0 [7:0]
7	7-0	ch1_Q0 [15:8]
8	7-0	ch0_I1 [7:0]
9	7-0	ch0_I1 [15:8]
10	7-0	ch0_Q1 [7:0]
11	7-0	ch0_Q1 [15:8]
12	7-0	ch1_I1 [7:0]
13	7-0	ch1_I1 [15:8]
...	...	...

is preferred since the samples can be represented into a smaller size. The data can be converted to a complex data type later for signal processing.

Information is communicated to and from the computer and the LimeSDR through packets. The total packet size is 4096 bytes and consists of two main parts: the header and the payload. The header is the first 16 bytes and contains the receiver and transmitter status flags, and the packet timestamp. The payload is the remaining 4080 bytes and contains the RF sample data. The format of the payload depends on the number of active channels and the sample data type. An example of the payload format is shown in table A.1 and table A.2. If there is one channel active, then the payload format would only include the singular channel's RF samples.

There is also the parameter "throughputVsLatency" for controlling configuration bias toward low latency (usually lower throughput) or high data throughput (usually results in higher latency). This value affects the size of data transfers from hardware.

A.4 Sampling Rate and Timestamps

The timestamp is a value of the hardware counter with the tick frequency the same as the sample rate. The receiver timestamp is when the first sample in the returned buffer was received. The transmit timestamp is when the first sample in the submitted buffer will be sent. A sample is a real part and a imaginary part.

The hardware counter is synced with the receiver sample rate so that the receiver stream needs to be activated in order for the hardware counter to work. There are currently no capabilities on the LimeSDR to set the timestamp for receiving. However, there is the capability of setting a timestamp for the transmitter. To synchronize the timestamps of the receiver and the transmitter, the transmitter timestamp can be offset of the receiver so that the timestamps will be aligned eventually when continually receiving.

Ultimately, even with the timestamp synchronized the hardware does not support perfect synchronization of the transmitter and receiver. However, the delay between the transmitter and the receiver can be considered consistent once the hardware is running. However, there may still be nanosecond variations, because the ADCs are not coherent with the carrier frequencies. The delay depends on the filter calibration, sampling frequency, and other parameters. It is possible to measure this delay and incorporate it into signal processing, effectively eliminating it.

The LimeSDR has the capability of oversampling in the hardware. If it is oversampled then the data is downsampled back to the specified sampling frequency before sending samples to the Raspberry Pi. A signal is said to be oversampled by a factor of N if it is sampled at N times the Nyquist rate. The oversampling factor can be set in the Lime Suite library.

A.5 Buffers

There are two types of buffers; one is in the software on the Raspberry Pi and the other is in the LimeSDR hardware FPGA FIFO. Each channel has its respective FIFO buffer and the FIFO size can be set. When receiving samples the FIFO continuously fills up (enqueue). To dequeue the receive FIFO, the software will call a chosen amount of samples to pull from the FIFO and transfer to the Raspberry Pi software buffer. This needs to be done frequently enough so the FIFO buffers do not overflow. Thus it is ideal to minimize processing time so that the processor can request

more samples from the FIFO more frequently. This can be done by using the LMS_FMT_I12 or LMS_FMT_I16 data type. Using these data types also increases the processing time when saving the data to a file. The receive FIFO is not capable of setting the timestamps. If the received samples are not required, the best option is just to keep receiving samples and throwing them away.

The transmit FIFO is filled up by the processor. The hardware will pull the samples of the FIFO and transmit them. The transmit hardware also has the capability of using timestamps so that it does not require continuous transmission. To verify the timestamps behave correctly, the transmit buffer needs to have samples in them before the timestamp occurs.

A.6 Fast Fourier Transform

A tool to use in signal processing is the discrete Fourier transform. One of the best options in C++ is the Fastest Fourier Transform in the West (FFTW) library. Since the Raspberry Pi OS is 32-bits and the LMS_FMT_F32 uses 32-bit floats, the FFTW library should also use 32-bit floats instead of 64-bit doubles. Certain libraries need to be linked when using the linker for this to happen as explained in Appendix B. The FFTW library uses the fftwf_complex data type. This data type is a two-dimensional (real and imaginary) array. It's important to note that for the same chunk of memory in C++ the following data types are the same: fftwf_complex bufferRX[N] = float bufferRX[N][2] = float bufferRX[2N]. Essentially they are all interchangeable and the I and Q samples are represented in memory the exact same.

A.7 Plotting

In order to plot data from the C++ code, gnuplot library is used. It is a well-documented library. The library is useful to plot the signals to visually understand the data being received. The gnuplot library was used extensively in debugging and is useful to see the f_{IF} signal.

A.8 Processing Via MATLAB

Included in the Thesis Code are MATLAB scripts used for processing the data. In order to capture more data faster without the FIFO buffer overfilling, the LMS_FMT_I12 data type is used. The software is configured to write the receive signal to a binary file. No signal processing

Table A.3: Links to various resources and tools used in the LFMCW radar.

Thesis Code: https://github.com/kohlsne/LimeSDR-LFMCW-Radar
Lime Suite Library: https://wiki.myriadrf.org/Lime_Suite
Lime Suite Library API https://docs.myriadrf.org/LMS_API/index.html
Lime Suite Example Code: https://github.com/myriadrf/LimeSuite/tree/master/src/examples
Lime Suite Example Code Documentation: https://github.com/myriadrf/LimeSuite/blob/master/docs/lms7_api_quick_start_guide.pdf
Stream Protocol: https://github.com/myriadrf/LimeSuite/blob/master/docs/StreamProtocol.pdf
FFTW Software: http://fftw.org
Gnu Plot: http://www.gnuplot.info

can be opted so that the software just transmits and writes the receive signal in a file. The receive signal saved in a file is offloaded from the Raspberry Pi onto a more powerful computer for post-processing. This can easily be done using the bash ”scp” command. The MATLAB script generates an identical transmit chirp that was used by the LimeSDR-Mini and the receive binary file can be read in and converted to a complex floating type identically to the LMS_FMT_F32 data type. Once this is done the transmit and receive signals can be mixed in MATLAB. The advantage of mixing the signal in MATLAB instead of in real-time while transmitting and receiving is the ability to run the radar for longer. Since the software is constantly removing samples from the FIFO more frequently without signal processing, the FIFO buffers do not overflow as fast or at all.

APPENDIX B. HOW TO COMPILE AND RUN THE SOFTWARE

This appendix lays out the steps required to run the code.

B.1 Hardware

The hardware used in this thesis:

- LimeSDR-USB or LimeSDR-Mini
- Transmit and receive antennas, with SMA connectors
- Raspberry Pi 4 computer
- Micro SD card (at least 32 GB)
- USB 3 female to male extension cable
- Coaxial cable to short the transmitter and receiver together (a couple centimeters long)
- Coaxial cable to measure the length
- Power Divider/ Combiner x2
- Attenuator 30 dB
- Laptop
- Power Source for Raspberry Pi
- Router
- Car inverter (or a battery with an inverter)
- Power extension cord

B.2 Raspberry Pi Control

The setup included the Raspberry Pi 4 connected to a router via Wi-Fi. This way from my laptop I can ssh onto the Raspberry Pi and also create a network mount to use an IDE on my laptop to configure files on the Raspberry Pi. In addition, VNC can also be enabled. If the Raspberry Pi does not have access to a router, the Raspberry Pi can create a Wi-Fi hotspot (internal network) so that a laptop can connect to it directly via Wi-Fi. The Raspberry Pi can use a script that will automatically create a hotspot when there is no recognized Wi-Fi network to connect to. This does not work as well as having a router to connect the host computer to the Raspberry Pi. Using a router to communicate with the radar system is probably the best option in an area without a network or is being flown around on a drone. The router does not need to be connected to the internet for the host computer to communicate with the Raspberry Pi. The Raspberry Pi itself can also be connected to a monitor, keyboard, and mouse.

B.3 Radar Setup

The best place to test the LFMCW radar is in an open area free of unwanted objects that can be detected. A football field, a large parking lot, or a park free of trees are good candidates. Often in these areas, it is difficult to find a power outlet to provide power to the Raspberry Pi. Using a car inverter with an extension cord to provide distance from the radar works well. Ideally, the car is parked far enough and in the null of the radiation pattern of the antennas, so that the car does not interfere with the characteristics of the antennas. The radar system is sensitive to the metal around it, so placing it on a wooden table is preferred over a metal one. Additionally, instead of a car inverter, a separate battery-powered inverter would also work. A router to have a laptop connect to the Raspberry Pi is the preferred method of controlling the Raspberry Pi. The loop antennas are sensitive to perturbations and objects around them. Ideally, a spectrum analyzer would frequency sweep the antennas and measure the VSWR. This would verify the antennas are efficient at the same frequencies and the carrier frequency on the LimeSDR-Mini can be configured to match the resonant frequency. However, this is impractical so the antennas should be characterized and then carefully handled.

Table B.1: Links to software tools used to configure the LimeSDR

PiSDR: https://pisdr.luigi.ltd
LimeSuite Library: https://wiki.myriadrf.org/Lime_Suite
Balena Etcher: https://www.balena.io/etcher
Automated Hotspot Script: https://www.raspberryconnect.com
Thesis Code: https://github.com/kohlsne/LimeSDR_LFMCW_Radar

B.4 Installing The Software

Installing all of the libraries onto the Raspberry Pi computer is time-consuming. I recommend using a modified Raspbian image with the latest SDR software preinstalled. Luigi Cruz created a Linux Distro known as PiSDR (see Table B.1 for links). Using this image provides a hassle-free way of having all the correct library packages installed on the Raspberry Pi. If the PiSDR image is no longer available it may be needed to download the necessary Lime Suite libraries on a Raspbian Image. To flash the image onto the micro SD card, I would recommend using the software Balena Etcher. In addition, the code used for this thesis can be pulled using git at my GitHub repository.

B.5 Downloading and Running the Code

- Find the IP address of the Raspberry Pi.
- Connect to the Raspberry Pi from a laptop and use the terminal command ”ssh pi@IP_ADDRESS“
- Enter password, the default password is ”raspberry“
- Run the command “git clone https://github.com/kohlsne/LimeSDR_LFMCW_Radar”
- change directory to the LimeSDR_LFMCW_Radar directory that was downloaded from the git repository. “cd LimeSDR_LFMCW_Radar”

- Compile the code: “g++ -o LFMCW main.cpp LFMCW.cpp -lLimeSuite -lfftw3f”
  - The -lfftw3f flag tell the linker to use floats (32-bits for the real part and 32-bits for the complex part of the sample) instead of doubles for the FFTW data type.
  - The -lLimeSuite flag tells the linker to link to the Lime Suite Library.
- Run the code: ./LFMCW “RXFileName.bin”
  - “RXFileName.bin” is the file that will be created that contains all of the RX samples in binary format
- Copy data to laptop: “scp pi@IP_ADDRESS:PATH_TO_RXFileName.bin .”
- Run the ProcessData.m MATLAB script from laptop.
  - This read in the “RXFileName.bin”’ file and converts the LMS_FMT_I12 data to complex floats.
  - The script also generates the TX signal and mixes it with the RX signal read in. It additionally performs a feed-through nulling technique and displays the resulting figures.