UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

DIGITAL PREDISTORTION OF WIDEBAND RADAR WAVEFORMS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

By

RANDALL SUMMERS
Norman, Oklahoma

2021

DIGITAL PREDISTORTION OF WIDEBAND RADAR WAVEFORMS

A THESIS APPROVED FOR THE
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY THE COMMITTEE CONSISTING OF

Acknowledgments

I want to thank my advisor, Dr. Mark Yeary, along with Dr. Hjalti Sigmarsson and Dr. Jay McDaniel for their support throughout this project. Their guidance over the last several years has been invaluable, and I have learnt a great deal from them over the course of this project. I would also like to thank my friends and family for all the moral support they've provided.

This work was partially supported by NASA Grant/Cooperative Agreement Number: 80NSSC18M0032 P00002. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect those of NASA.

Table of Contents

List of Tables

List of Figures

Abstract

As demands for increased radio frequency system performance have grown over time, sufficiently mitigating the distortion introduced by high power amplifiers has proven to be a major challenge to overcome. Digital predistortion is one technique commonly used to reduce the distortion products seen at the output of amplifiers, to great effect. While some of the most widely used predistortion algorithms to date may seem most applicable to waveforms with properties most commonly seen in wireless communication systems, it will be seen that with some specialization, these methods can be quite applicable to linear frequency-modulated pulse radar waveforms. In this thesis, a frequency-domain approach to parameter estimation is presented, and a simple window-based approach to predistortion of wideband linear frequency-modulated pulses is also explored. To test the performance of these predistortion algorithms, a modular radar signal processing FPGA testbench architecture is discussed. Finally, a software defined radio was used to measure the performance of the algorithms discussed in this thesis, and a significant improvement in the peak range sidelobe level was observed. These results enable the integration of digital predistortion into modern radar systems with minimal computational overhead to correct radar pulse envelope distortion.

Chapter 1

Introduction

In recent decades, the demand for capacity of wireless communications systems has grown tremendously, and future projections indicate that this growth will continue. The proposed trends in increased data rates range from a growth of 10$\times$ to 1000$\times$ every 5-years [1,2]. As modulation schemes and data processing capabilities keep increasing, the radio-frequency hardware presents a bottleneck due to inherent bandwidth limitations, nonlinearities, and in general the cost, size, weight, and power consumption (C-SWaP) of components and systems. High-power amplifiers (HPAs) are key components for the functionality of the transmitter portion of any system. The power consumed by the HPA is generally the bulk of the overall system power consumption [3]. Therefore, optimizing amplifier power efficiency has been a very active research topic. Multiple different amplifier configurations have been proposed to improve the power efficiency. However, to date, most commercially available HPAs are operating with less than 50% efficiency. Traditionally, the maximum efficiency of a power amplifier is reached when the amplifier is operated at peak power [4]. The drawback of this operation is that it results in the introduction of non-linear behavior to the system. However, with the increased processing flexibility of modern digital transceivers, the nonlinearities can be mitigated by using digital predistortion [5].

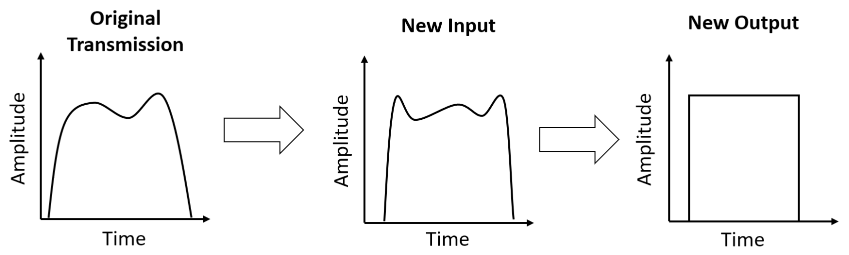


Figure 1.1: Illustration of the basic predistortion task investigated in this work.

Digital predistortion works by attempting to apply an inverse distortion to a digital signal before being sent through a DAC and the remainder of the RF chain that corrects for the nonlinearities present in the system, resulting in the desired signal ultimately appearing at the system output, as illustrated in Figure 1.1. [6–10]. As much of the demand for predistortion has historically stemmed from the telecommunications industry, much of the research into predistortion algorithms has been targeted towards communications applications and its particular set of system requirements. One common aspect of many existing predistortion algorithms is that they estimate the predistortion coefficients using techniques based upon time-domain analysis. This has been done for a variety of reasons, not the least of which being relatively low computational complexity. This approach is not without its challenges, however. To accurately estimate the parameters using time-domain techniques, it is required that the received signal be precisely aligned with the corresponding samples of the transmitted signal. Additionally, many predistortion techniques are designed to correct the system response over a range of envelope levels, an aspect that may be useful for signals with varying amplitude, but is less useful when working with a linear frequency modulated (LFM) pulsed radar system in which the envelope voltage is constant (only) when transmitting.

For an LFM pulse radar system, it should therefore be apparent that there are advantages to exploiting the structure of the transmit waveform to develop a predistortion system with some tolerance for misalignment of the transmitted and received waveforms and avoid computing predistortion coefficients for envelope amplitudes that will not be used due to the constant-amplitude nature of the waveform [11]. As such, this thesis will focus on the design of predistortion algorithms that exploit the properties of LFM radar waveforms to achieve improved performance while minimizing computational complexity and numerical instability.

Of course, it is necessary to test the performance of a predistortion algorithm prior to integration into a full radar system. To facilitate this process, two main approaches were used. Firstly, the implementation of these predistortion algorithms on an FPGA was investigated [12], as an FPGA is the logical place to implement such algorithms in many modern radar systems such as SESAR [13, 14] and EcoSAR [15, 16]. As is well known, there are three primary types of digital receiver technologies: heterodyne, direct sampling, and direct conversion. The FPGA architecture described in this thesis is focused on direct sampling (i.e. the analog-to-digital converter (ADC) is connected to the antenna with no analog frequency translation circuits in between). With the advent of extremely high-speed ADCs that offer a sufficient number of bits of resolution [17–19], direct sampling is possible for radar systems that operate in the lower RF bands. For example, time interleaved ADCs with low spurs are one of the technologies that have significantly changed the landscape of ADC capabilities over the last ten years or so (see [20–22]). This architecture study is specifically devoted to a system that operates in the P-band (specifically with a center frequency of 435 MHz), which is ideal for direct conversion with modern data converters. Many details of this system are discussed in a previous paper [14]. The FPGA architecture study contained in this thesis concentrates on the firmware interface that is required to unite one of these types of ADCs with a state-of-the-art field-programmable gate array (FPGA).

Specifically, that portion of the thesis concentrates on creating a firmware interface to data converters that is useful for the prototyping of signal processing algorithms. In the course of developing radar signal processing algorithms, it is frequently useful to be able to generate and measure radar waveforms on a hardware platform as closely resembling the target system as possible. At the same time, it is generally desirable for the system to be easily reconfigurable to enable quick testing of various system and radar waveform parameters. In a modern context, this would generally mean being able to easily reconfigure the testbench from a user's general-purpose computer. Secondary goals for such a radar testbench include leveraging existing intellectual property (IP) to lower time-to-market and increase reliability, and a modular design to reduce barriers to changes in the underlying hardware architecture. In this thesis, one possible system architecture is proposed and discussed for a high-bandwidth radar testbench.

While an FPGA is a practical place to implement real-time predistortion in many modern radar systems, it is arguable that designing custom digital hardware to implement predistortion algorithms is not an ideal approach for initial verification of predistortion algorithms due to a relatively slow development cycle. Compared to digital hardware design, implementing predistortion algorithms using computer software can be expected to usually offer a much quicker development cycle. As such, the use of a software defined radio (SDR) for verification of predistortion algorithms was also investigated for this work.

Chapter 2 presents background information that is used to motivate the development of two predistortion algorithms tailored to linear frequency modulated pulse radar waveforms in Chapter 3. Chapter 4 discusses various aspects of the design of a modular FPGA-based radar DSP algorithm testbench, and Chapter 5 discusses the design and implementation of an SDR testbench which was used to test the algorithms presented in Chapter 3. Finally, Chapter 6 presents the results of these measurements and Chapter 7 concludes this thesis.

Chapter 2

Background

This chapter will focus on commonly used predistortion techniques, and some inherent limitations and requirements. Additionally, this chapter will introduce some figures of merit by which radar waveform predistortion algorithms can be judged.

2.1 Fundamentals of Digital Predistortion

At its core, predistortion is basically the task of modelling the distortion introduced by a system, and designing a new inverse system that mitigates the overall system distortion when integrated with the original system. As a result, it is crucial to understand the way distortion is often modelled in RF systems. As has been discussed, RF power amplifiers are one of the largest contributors of distortion in RF systems in many cases, and as a result much effort has been invested in finding ways to model the distortion introduced by RF amplifiers. While there is much interest in high efficiency amplifier topologies such as envelope tracking amplifiers and switch mode power amplifiers, linear power amplifier architectures remain the prototypical choice in RF systems. As a result, much of the literature concerning digital predistortion is centered around the distortion introduced by linear power amplifiers.

While there are many models of linear power amplifier distortion that have been investigated, one of the most accurate is the discrete Volterra series

$$y[n] = \sum_{i=0}^{M} h_1[i]x[n-i] \\ + \sum_{p=2}^{P}\sum_{i_1=0}^{M}\cdots\sum_{i_{2p-1}=0}^{M} h_{2p-1}(i_1,\cdots,i_{2p-1})\prod_{j=1}^{p} x[n-i_j]\prod_{k=p+1}^{2p-1} x^*[n-i_k].$$

This model is quite general, but it also has a few drawbacks. First, the number of coefficients to estimate quickly explode as one attempts to model higher order distortion. Additionally, some of the Volterra coefficients are coupled to each other, complicating the estimation of model coefficients. As a result, many simplified versions of the Volterra series have been developed to attempt to ameliorate these issues [23] [24].

2.2 The Memory Polynomial Model

Of the various methods of pruning the Volterra series, one of the most popular is the memory polynomial model [6, 7, 25]

$$y_{MP}(n) = \sum_{k=0}^{K-1}\sum_{m=0}^{M-1} a_{km}x(n-m)|x(n-m)|^k \tag{2.1}$$

where x is the input to the model, y_{MP} is the corresponding output, K is the envelope order, M is the memory order, and the a_{km} are the model coefficients. This model has proven itself useful for robustly modeling and compensating for the nonlinearities introduced by a wide variety of power amplifiers [6, 7, 25]. One way to look at this model is as a bank of finite impulse response (FIR) filters, the input of

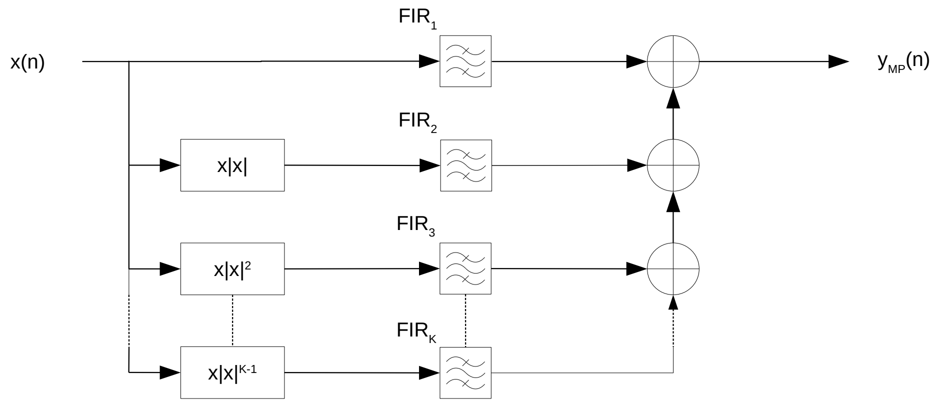


Figure 2.1: Illustration of the memory polynomial model. Reprinted from Summers et al. (2020) © 2020 IEEE.

each separately weighted by the signal envelope's magnitude, as shown in Figure 2.1.

One nice aspect of this model is the ease with which the model coefficients can be estimated. Since the model is linear with respect to the coefficients a_{km}, the coefficients can be estimated using a simple least-squares process described in [7]. It should be noted that this process depends on comparing the discrepancy between the transmitted waveform and the waveform that was supposed to be transmitted in the time domain. As such, acquisition of these sample-synchronized waveforms generally requires some level of time synchronization of the transmitter and receiver, however the final alignment of the waveforms can usually be accomplished by cross correlation of the two signals, to account for the delay introduced by the analog signal chain of the transceiver.

Another interesting feature of the coefficient estimation process is the relationship between the waveforms used to estimate memory polynomial coefficients and the quality of coefficient estimation. It is commonly accepted wisdom that the best quality of coefficient estimation occurs when the signals used for coefficient estimation resemble the signals that one ultimately wants to transmit. This may not always be the case, however, since some signal varieties do not lend themselves particularly well towards memory polynomial estimation. Specifically, it can be seen from Equation 2.1 or Figure 2.1 that signals with little amplitude modulation might result in matrices during the estimation process that are ill conditioned, and possibly lead to erroneous results.

Finally, it should be noted that when predistortion is used to reduce spurious emissions, the predistortion system's ADC and DAC need to sample fast enough to observe and correct for the spurious emissions, respectively [5]. As such, it is unreasonable to expect predistortion to directly correct for spurious emissions when applied to signals with a bandwidth that is large relative to the sampling rate of the system's DACs and ADCs.

2.3 Radar Waveform Performance Metrics

In order to judge the performance of a predistortion algorithm in the context of a radar system, it is clear that the impact of the algorithm on the transmitted waveform's performance metrics should be inspected. In this thesis, four main figures of merit are considered, namely the range resolution, effective bandwidth, peak sidelobe level, and integrated sidelobe ratio. This section provides a brief description of these metrics.

2.3.1 Range Resolution

As the acronym for radar suggests, one of the main functions of a radar system is to measure the range to a target. As such, the ability of a radar to distinguish adjacent targets is of much importance. While there are multiple considerations that impact the ability of a radar to distinguish adjacent targets, one of the most obvious is the width of a radar pulse after pulse compression. This width is called the range resolution of the radar, and throughout the remainder of this thesis will be measured $3\,\mathrm{dB}$ below the peak of the pulse compressed pulse. The range resolution is nominally given by the equation

$$\Delta R = \frac{c}{2B} \tag{2.2}$$

where c is the wave propagation velocity (usually in free space) and B is the bandwidth of the radar waveform, however it must be calculated using numerical techniques when it is desired to find the range resolution of a measured pulse.

2.3.2 Effective Bandwidth

The effective bandwidth of a waveform is the quantity β defined by the equation

$$\beta^2 = \frac{\int_{-\infty}^{\infty} (2\pi f)^2 |S(f)|^2 \, \mathrm{d}f}{\int_{-\infty}^{\infty} |S(f)|^2 \, \mathrm{d}f} \tag{2.3}$$

where $S(f)$ is the Fourier transform of the waveform in question [26]. From this definition, it can be seen that the effective bandwidth is the root-mean-square deviation of the spectral power of the waveform from the mean (center) frequency. This quantity is of interest due to being directly related to the accuracy of time delay measurements in radar systems. Specifically, the root-mean-square error of time-delay measurements, δT_R is given by

$$\delta T_R = \frac{1}{\beta (2E/N_0)^{1/2}} \tag{2.4}$$

where E is the energy of the radar pulse and N_0 is the noise power per unit bandwidth [26].

2.3.3 Sidelobe Level Measurements

Yet another important aspect of a radar waveform is the height of the range sidelobes. The peak sidelobe level is a fairly straightforward metric, being the peak value of the pulsed compressed waveform outside the main lobe. While the peak sidelobes will be the first to cause false target detection by strong targets, the total energy contained in the sidelobes is also an important factor that impacts the quality of radar measurements. This can be characterized by the ratio of the waveform energy contained in the sidelobes to the energy contained in the main lobe, also known as the integrated sidelobe ratio which is given by

$$\text{ISLR} = 10\log_{10}\frac{\int_{-\infty}^{a}|x(\tau)|^2\,\mathrm{d}\tau + \int_{b}^{\infty}|x(\tau)|^2\,\mathrm{d}tau}{\int_{a}^{b}|x(\tau)|^2\,\mathrm{d}tau} \tag{2.5}$$

where $x(\tau)$ is the pulse compressed waveform, and a and b are the locations of the beginning and end of the main lobe respectively.

This chapter has helped to explain the Volterra series and the memory polynomial model in the context of digital predistortion. Additionally, several figures of merit by which radar waveform predistortion algorithms can be judged. The next chapter uses this background to define an optimization problem for determining filter coefficients using the Chebyshev approximation.

Chapter 3

Predistortion of Wideband Radar Systems

In this chapter, the background information from Chapter 2 is used to motivate the development of two predistortion algorithms applicable to wideband pulse-Doppler radar systems. Both algorithms exploit the Fourier transform to estimate system distortion, but the method of modifying the radar pulse to account for system distortion differs. The first algorithm designs an optimal Chebyshev filter to correct system distortion, while the second algorithm uses knowledge of the pulse's instantaneous frequency to design a window function that compensates for system distortion.

3.1 Predistortion Using an Optimal Chebyshev Filter

As was mentioned in Chapter 2, the memory polynomial model provides a great deal of flexibility for the predistortion of systems with a wide variety of input signal statistics and nonlinear characteristics. This flexibility is however not an ideal use of computational resources in an LFM pulse radar system with a mostly constant signal amplitude. By setting the maximum envelope order K to 1, the memory polynomial model reduces to a single FIR filter, indicating that a FIR filter with appropriately chosen taps should be able to perform well for predistorting a system with a constant amplitude signal.

Many implementations of the memory polynomial model rely on methods based on least-squares optimization in the time-domain [7, 27]. This works well when the measured output waveform samples can be precisely aligned with the input waveform samples, but this can be surprisingly difficult to do without designing a system with this objective in mind. On the other hand, it is almost trivial to align the output of an LFM pulse radar system and the input in the frequency domain. To avoid the issues surrounding alignment of transmitted and received waveforms in time, it follows that the system transfer function can be estimated at a discrete number of frequencies by taking the fast Fourier transform (FFT) of both transmitted and received waveform and taking the ratio

$$H(\omega_k) = \frac{Y(\omega_k)}{X(\omega_k)} \qquad k = 1, ..., M \tag{3.1}$$

where the ω_k are the frequencies at which it is desired to optimize the frequency response of the system. It should be noted that designing an equalizer within only a narrow frequency band can sometimes result in undesirable effects on the edges of a waveform. One strategy to combat this is to extrapolate the frequency response outside the waveform bandwidth with linear phase, as is done for the simulations presented in this paper. Given the frequency response of the system at a discrete number of frequencies, an FIR filter can then be designed to approximate a desired response.

If a flat, linear-phase frequency response is desired, a logical choice for the target frequency response is a delay line

$$H_{des}(\omega_k) = e^{-jD\omega_k} \tag{3.2}$$

where $D \in \mathbb{R}_+$ is chosen so that the system is not attempting to create a smaller delay than is present without the filter in the system. If the frequency response of the FIR filter is denoted $H_{fir}(\omega)$, the overall system frequency response is given by

$$H_{cas}(\omega_k) = H_{fir}(\omega_k)H(\omega_k) \tag{3.3}$$

It then remains to define the optimization problem that will be solved to determine the FIR coefficients. One good choice is a variation of the Chebyshev approximation problem:

$$\text{minimize} \max_{k=1,...,M} \|H_{cas}(\omega_k) - H_{des}(\omega_k)\|_2 \tag{3.4}$$

where $\|\cdot\|_2$ is the Euclidean norm. This problem can be rewritten as

$$\begin{array}{rl} \text{minimize} & t \\ \text{subject to} & \|H_{cas}(\omega_k) - H_{des}(\omega_k)\|_2 \leq t, \\ & k = 1, ..., M \end{array} \tag{3.5}$$

Because the frequency response of an N-tap FIR filter with taps $h(n)$ is given by

$$H_{fir}(\omega_k) = \sum_{n=0}^{N-1} h(n)e^{-j\omega_k n} \tag{3.6}$$

or equivalently

$$\begin{bmatrix} \mathrm{Re}\{H_{fir}(\omega_k)\} \\ \mathrm{Im}\{H_{fir}(\omega_k)\} \end{bmatrix} = B * \begin{bmatrix} \mathrm{Re}\{h(0)\} \\ \mathrm{Re}\{h(1)\} \\ \vdots \\ \mathrm{Re}\{h(N-1)\} \\ \mathrm{Im}\{h(0)\} \\ \mathrm{Im}\{h(1)\} \\ \vdots \\ \mathrm{Im}\{h(N-1)\} \end{bmatrix} \tag{3.7}$$

where B is given by

$$\begin{bmatrix} 1 & \cos(\omega_k) & \dots & \cos((N-1)\omega_k) & 0 & \sin(\omega_k) & \dots & \sin((N-1)\omega_k) \\ 0 & -\sin(\omega_k) & \dots & -\sin((N-1)\omega_k) & 1 & \cos(\omega_k) & \dots & \cos((N-1)\omega_k) \end{bmatrix} \tag{3.8}$$

the frequency response of the uncorrected system can be represented in matrix form as

$$H(\omega_k) = \begin{bmatrix} \mathrm{Re}\{H(\omega_k)\} & -\mathrm{Im}\{H(\omega_k)\} \\ \mathrm{Im}\{H(\omega_k)\} & \mathrm{Re}\{H(\omega_k)\} \end{bmatrix} \tag{3.9}$$

and the desired frequency response can be represented as

$$H_{des}(\omega_k) = \begin{bmatrix} \mathrm{Re}\{H_{des}(\omega_k)\} \\ \mathrm{Im}\{H_{des}(\omega_k)\} \end{bmatrix} \tag{3.10}$$

the optimization problem can be rewritten

$$\begin{aligned} &\text{minimize} && t \\ &\text{subject to} && \|A_k h - b_k\|_2 \le t, \qquad k = 1, ..., m \end{aligned} \tag{3.11}$$

which can be recognized as a canonical second-order cone program, a subset of convex optimization problems, in h and t, and can therefore be solved using a variety of standard methods such as the interior point method [28–30].

3.2 Predistortion Using Transfer Function Windowing

One nice property of linear frequency modulated chirps is the direct relationship between time and instantaneous frequency. Since the optimal filter design approach above can only be expected to correct for first-order distortion products, it seems reasonable to diverge from the above approach after smoothing the estimated transfer function, and instead of designing a filter to predistort the waveform, design a windowing function to predistort the waveform using the correspondence between instantaneous frequency and time. Specifically, at each point in time t of the pulse, the window is defined such that

$$w(t) = \frac{1}{|H(\omega(t))|}. \tag{3.12}$$

where $\omega(t)$ is the instantaneous frequency of the chirp at time t. While this approach does not have as much theoretical background to support it as linear filter design, its performance seems comparable to the above approach, as will be seen in Chapter 6.

This chapter has discussed a convex optimization problem for finding filter coefficients for predistorting wideband radar waveforms. Additionally, a windowing approach to create predistorted radar pulses is introduced. The next chapter will discuss the design of a modular FPGA-based radar DSP algorithm testbench for the validation of predistortion algorithms.

Chapter 4

FPGA Architecture Study

In the context of a full radar system, it would be desirable for the predistortion algorithms discussed in Chapter 3 to run in real-time. To achieve this in many modern radar architectures, it makes the most sense to implement the predistortion algorithms on the FPGA controlling the DACs and ADCs of the radar system. Such an implementation demands the consideration of a multitude of factors. As a result, an architecture study discussing how one might go about implementing real-time predistortion on an FPGA is presented in this chapter.

4.1 Architecture overview

Many high-bandwidth radar platforms, such as NASA's Space Exploration Synthetic Aperture Radar (SESAR) [13, 14] and Ecological Synthetic Aperture Radar (EcoSAR) [15, 16], rely on FPGAs for low-level control of their data converters, due to the high data rates and precise timing required by the system. Similarly, an FPGA forms the heart of the proposed testbench, both to satisfy the same system requirements present in full radar systems, and because performing digital signal processing (DSP) on a system's FPGA is an exciting avenue of research which the proposed testbench seeks to enable. The major hardware components in the proposed testbench are then connected as shown in Fig. 4.1. The analog sub-assembly contains the filtering and amplification components required by the system, but does not contain any frequency-translation components.

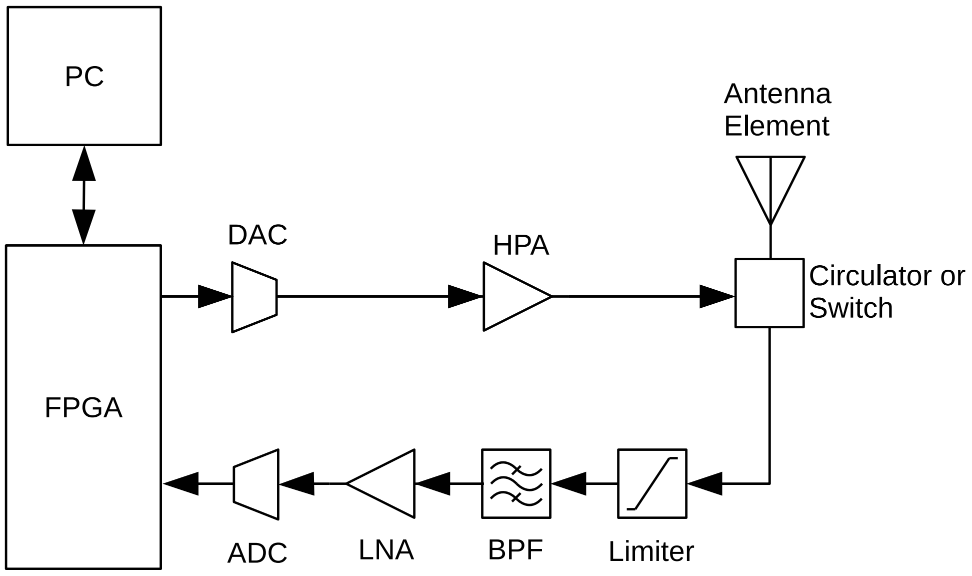


Figure 4.1: Block diagram of the main physical hardware components in a radar test bench. Reprinted from Summers et al. (2021) © 2021 IEEE.

As data rates on modern data converters can exceed $1\,\mathrm{GSPS}$ [31], it is unreasonable for the testbench to stream raw transmit and receive waveforms to and from the personal computer (PC) in real-time. At the same time, the raw waveforms are of significant interest in the development of DSP algorithms. To ameliorate this issue, the controlling PC can send a waveform to the FPGA to be stored in memory and transmitted repeatedly at some desired pulse repetition frequency. In the other direction, the PC can send a request to the FPGA for the received waveform to be recorded to memory for some specified length of time and then sent to the PC at a slower data rate more appropriate for the peripherals commonly available.

4.1.1 Internal FPGA Architecture

Up to this point, the FPGA has been discussed as a black box, however, the design of the FPGA hardware is clearly crucial to the functionality of the proposed testbench. One of the most important design choices is the mechanism by which the PC and FPGA will communicate. A wide variety of options exist for communication – for example, a universal asynchronous receiver-transmitter (UART) interface [32, 33] could be implemented on the FPGA to directly read and write to and from a bank of control registers, and data could be transferred directly to/from banks of waveform RAM. This approach suffers from a number of disadvantages. First, UART is relatively slow and would make the transfer of large waveforms cumbersome. Secondly, while this approach is relatively simple at first, as system complexity increases, the control interface becomes complex and difficult to modify.

Another popular approach is to implement a soft processor core ("PS") on the FPGA which handles communications between the PC and FPGA along with a variety of configuration and control tasks within the FPGA. The utilization of processor cores within the FPGA considerably simplifies the implementation of more complex communication and control schemes, and empirically seems to be a common design choice in software-defined radios (SDRs). This soft processor core can run a bare-metal program (i.e. a program running on the processor without an operating system), or a full Linux operating system. Both options provide distinct advantages. On one hand, bare-metal systems are somewhat simpler to implement, with a more direct interface to underlying hardware systems. Bare-metal programs also tend to run faster without the overhead of an operating system [34–36]. At the same time, a full Linux operating system allows the use of robust and widely used libraries such as the Linux networking stack and FFTW (a popular fast Fourier transform library) [37]. As a result, one might implement a bare-metal system during the initial implementation of system drivers, and later run an embedded Linux distribution to enable higher levels of abstraction and code reuse.

During initial driver development, the simplest route is likely to utilize a UART transceiver for all PC-FPGA interaction. Later in the testbench development cycle, a full Linux operating system would likely be more desirable, so the UART will be utilized to provide a serial console with which the system can be controlled through the Linux kernel. Due to the text-based nature of the Linux console necessitating compatibility layers such as XMODEM to transfer binary data, along with slow communication rates, the UART is likely not optimal to transfer waveform recordings in the later testbench development stages. Rather, the FPGA's Ethernet interface can be connected to the PS core after initial driver development, enabling faster binary data transfer in addition to the option of configuring the FPGA over the network. As the PS will run Linux, the kernel's networking libraries can be utilized for robust and abstract network programming.

4.1.1.1 Networking Stack

It is desirable for the proposed testbench to interact with the PC over Ethernet as much as possible. While many networked SDRs use the User Datagram Protocol (UDP) for data streaming, the non-real-time nature of the proposed testbench means reliable data transmission is needed. As such, it is proposed that TCP be used for the transport layer. At the application layer, the Hypertext Transfer Protocol (HTTP) provides a natural interface for the desired testbench behavior, as the PC can easily post waveforms and configurations to the FPGA and send GET requests for measurements. With a central processing unit (CPU) on both ends of the communication link, JavaScript Object Notation (JSON) would provide a convenient, human-readable configuration format for the proposed testbench, and the waveforms can be transported as "application/octet-stream" binary data files. Table 4.1 shows the proposed networking stack for sending/receiving data and controlling the FPGA.

Table 4.1: Networking stack for sending and receiving data to and from the FPGA. Reprinted from Summers et al. (2021) © 2021 IEEE.

Client application	PC
HTTP, TCP/IP libraries	
Host OS drivers	
Ethernet hardware	
Ethernet	Cable
Ethernet hardware	FPGA
FPGA PS core	
Embedded Linux drivers	
HTTP, TCP/IP libraries	
DAQ Application	
Data converter interface	
Analog sub-assembly	Analog hardware

4.1.1.2 FPGA-Data Converter Interface

Another important architectural choice is where to store transmit and receive waveforms and how to move data between the storage location and data converters. It is relatively straightforward to store and retrieve data in an FPGA's block RAM, however, these resources are typically limited. As it must handle large waveforms, it will be preferable for the proposed testbench to store waveforms in external double data rate synchronous dynamic random-access memory (DDR RAM). This choice, however, complicates the storage and retrieval of data. A DDR memory converter needs to be implemented along with interfaces to pull data from RAM and send a data stream to the testbench's digital-to-analog converter (DAC), and to write the testbench's ADC data stream to memory. The accepted solution for such problems in Xilinx FPGAs seems to be the use of an Advanced eXtensible Interface direct memory access (AXI DMA) core [38] which creates AXI-stream [39] interfaces to RAM. It is worth noting that Xilinx does not seem to provide Linux drivers for their DMA IP useable in user space. There do exist, however, some open source projects to provide user space DMA drivers such as libaxidma [40]. An illustration of the architecture is shown in Fig. 4.2.

The ADC and DAC data converters relevant to the testbench utilize low-voltage differential signaling (LVDS) interfaces that are characterized by the two timing diagrams that are illustrated in Fig. 4.3. Each interface transfers data between the data converters and FPGA. However, there is no reason a similar architecture could not be used for data converters with different interfaces such as JESD204B. Regardless, the remaining task is to convert the data streams to a format appropriate for the data converter interfaces. If desired, real-time processing could be implemented in hardware at this stage, for example filtering or mixing operations. Finally, most data converters require setting control registers on startup to operate as desired. This can easily be achieved by connecting the relevant serial interfaces to the PS, which can then run the desired configuration steps whenever desired.

Fig. 4.3(a) deals with this project's ADC. The converter has two data channels that send a new sample to the FPGA on each transition of DACLKP/N and DBCLKP/N [41]. In brief, DACLKP and DBCLKP are differential clocks that the ADC sends to the FPGA to coordinate timing with data samples. Fig. 4.3(b) is related to this project's DAC. The converter has four I/Q channels that are sent updated values through D[15:0]P/N. The data for each channel (A, B, C, and D) is sent to the data converter sequentially on each transition of DATACLKP/N before the next sample is received. The data converter and FPGA synchronize the samples

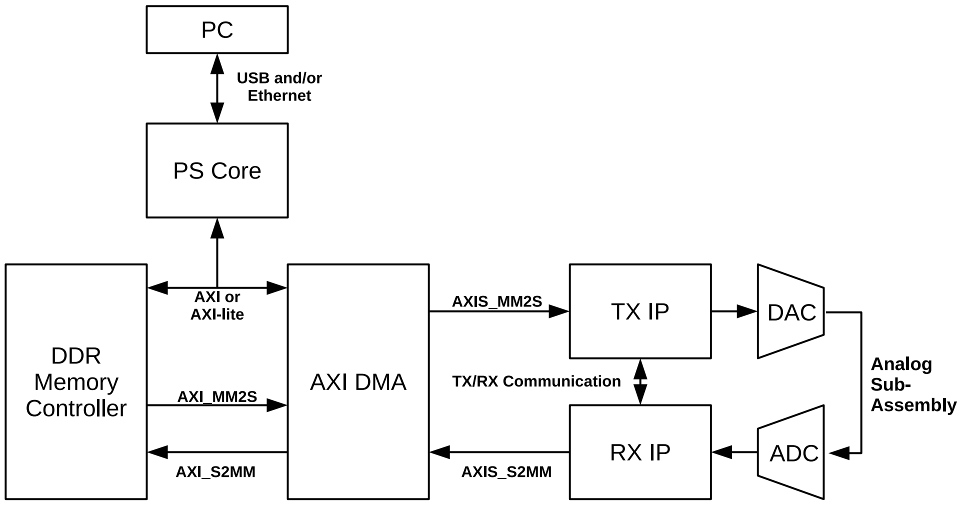


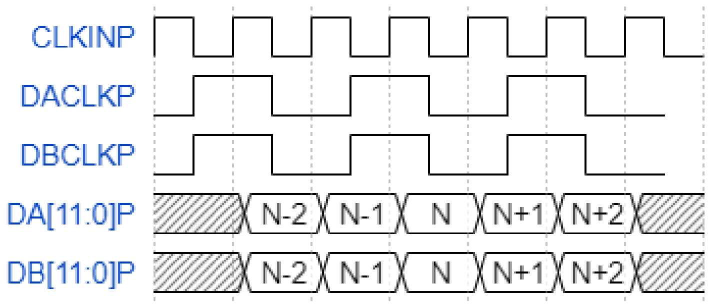
Figure 4.2: Block diagram of the FPGA system discussed for transmitting and receiving radar waveforms. Reprinted from Summers et al. (2021) © 2021 IEEE.

and channel numbers through occasional transitions of SYNCP/N [31].

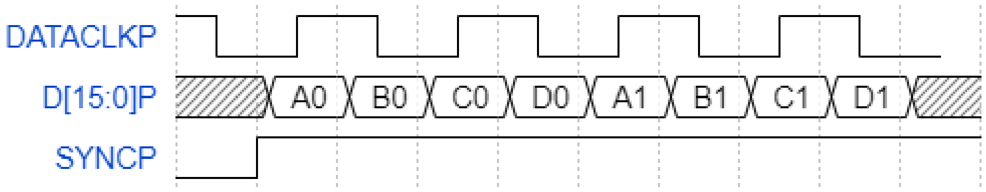
4.1.2 Current Results

Work has been done to verify this testbench architecture with a Virtex 7 FPGA and two high-speed data converters. The resource utilization of the latest iteration of the testbench is represented in Table 4.2 and Fig. 4.4. The specifications of these resources are discussed in depth in [42,43], a summary of which is provided below:

- LUT are 6-input lookup tables.
- LUTRAM are distributed synchronous RAM resources suitable for small arrays with low latency requirements.
- FF are flip-flops.
- BRAM are block RAM resources, suitable for larger arrays than LUTRAM.



(a) LVDS Interface for the ADS5402 ADC.



(b) LVDS Interface for the DAC3484 DAC.

Figure 4.3: Timing diagrams of the LVDS data transfer interfaces for the DAC and ADC used in this project. Reprinted from Summers et al. (2021) © 2021 IEEE.

- DSP are "DSP48E1 slices" with dedicated hardware for various common DSP functions such as multiplication, accumulation, and bit shifting.
- IO are the input and output pins.
- BUFG are global clock buffer resources.
- MMCM are mixed-mode clock managers.
- PLL are phase-locked loops.

This chapter has studied many aspects relevant to the design of a modular FPGA-based radar DSP algorithm testbench. The next chapter will discuss the design and implementation of a software defined radio testbench that enables comparatively rapid iteration in the design and validation of predistortion algorithms.

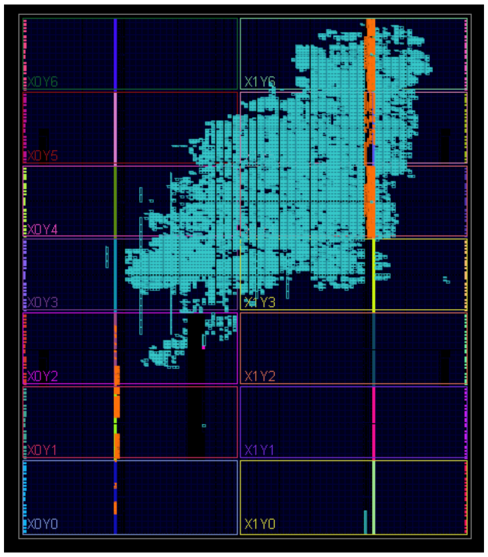


Figure 4.4: Current floorplan of the implemented FPGA design. Reprinted from Summers et al. (2021) © 2021 IEEE.

Table 4.2: Current resource utilization of the implemented FPGA design. Reprinted from Summers et al. (2021) © 2021 IEEE.

Resource	Utilization	Available	Utilization %
LUT	25894	303600	8.53
LUTRAM	6005	130800	4.59
FF	26824	607200	4.42
BRAM	70	1030	6.80
DSP	4	2800	0.14
IO	181	700	25.86
BUFG	6	32	18.75
MMCM	2	14	14.29
PLL	1	14	7.14

Chapter 5

SDR Testbench

To measure the performance of the digital predistortion algorithms designed for wideband radar waveforms described in Chapter 3, an SDR based testbench was implemented using an Ettus X310 software defined radio. The radio was connected to the testbench's host PC using a 10 Gbit Ethernet interface, enabling sampling rates of up to 200 MSPS at complex baseband. The particular radio used contained an Ettus SBX400-4400 daughtercard, which allowed for a maximum signal bandwidth of 120 MHz due to limitations of the analog components. To observe the transmitted signal, the transmit port of channel 1 of the SDR was looped back to the receive port of channel 1, with a 30 dB attenuator added in between to keep the power input to the receive port below the maximum acceptable level. While there were no external power amplifiers in the testbench, the discussion from Chapters 2 and 3 is still relevant due to the amplification elements internal to the SDR. The physical setup of the testbench used is shown in Figure 5.1.

The software portion of the testbench relies on the SoapySDR SDR support library to interface with the SDR hardware. The testbench's control flow is conceptually simple, repeatedly observing the system's input and output and designing new waveforms accordingly. The testbench starts with an ideal windowed chirp, and observes the waveform that is actually transmitted. Then, one of the algorithms

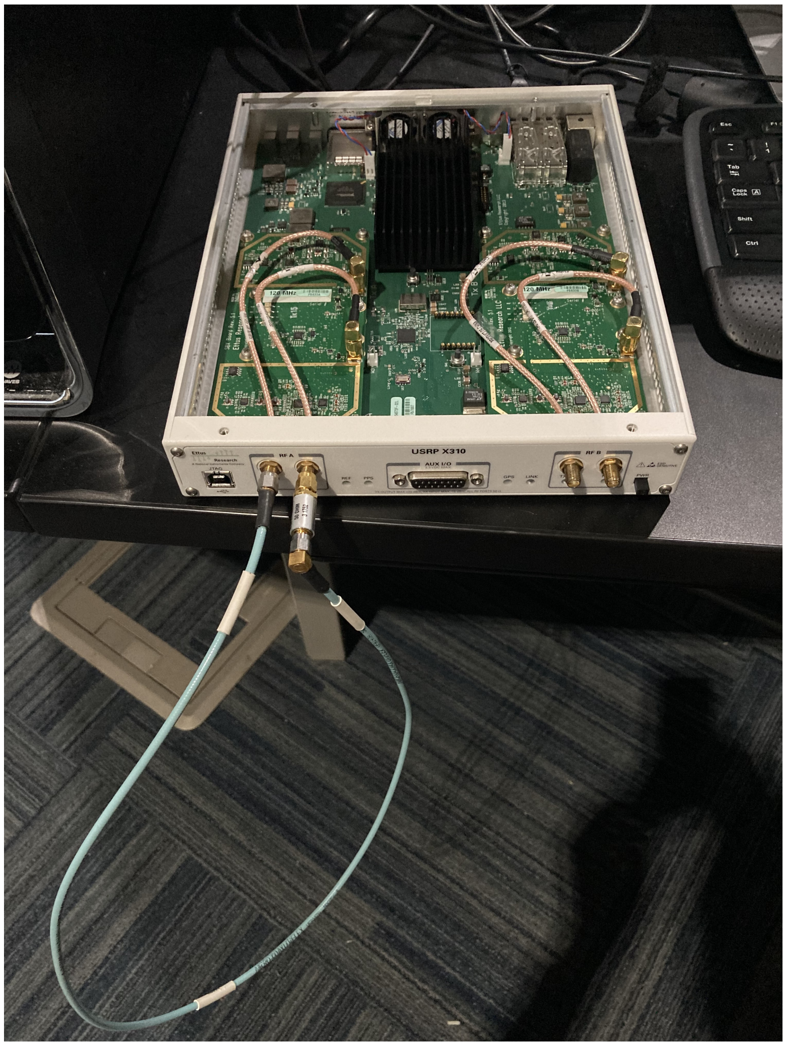


Figure 5.1: Physical setup of the software defined radio testbench.

from Chapter 3 is applied, creating a predistorted waveform that should result in a waveform closer to the ideal chirp being transmitted than before. The transmitted signal is once again observed, a new predistorted signal is generated, and the cycle repeats as such until the testbench has run as long as desired. This process is illustrated in Figure 5.2.

As the goal of this testbench is to test the usefulness of the predistortion algorithms of Chapter 3 for P-band radar systems with a relatively large percentage

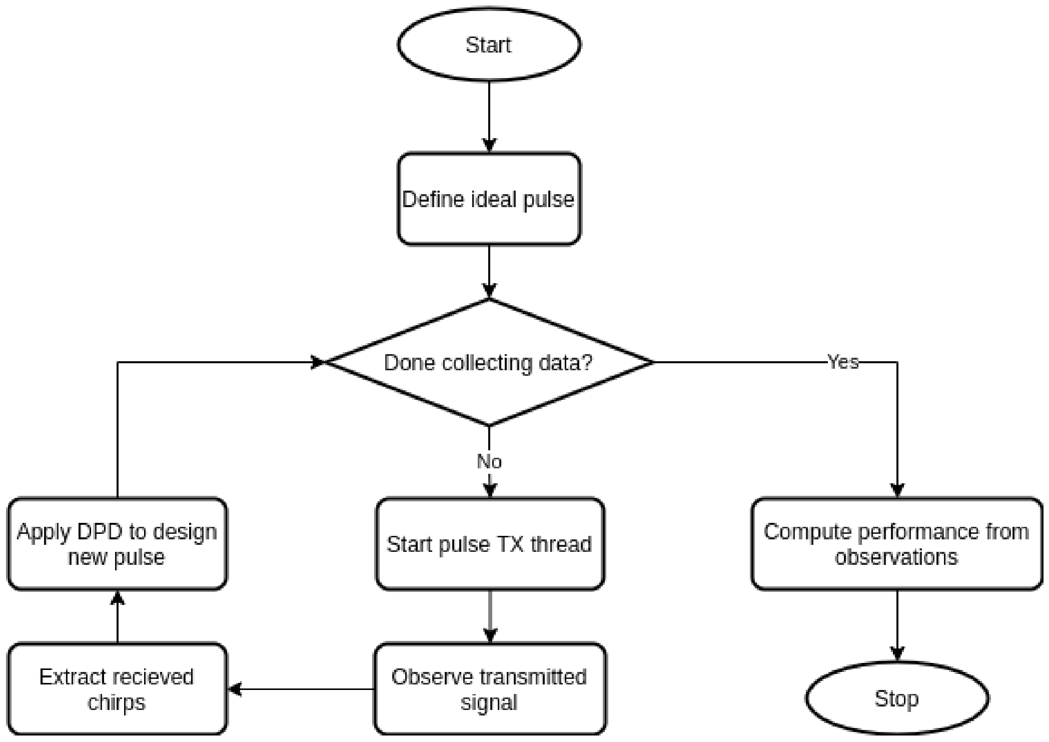


Figure 5.2: Flow diagram illustrating basic function of the software defined radio DPD testbench.

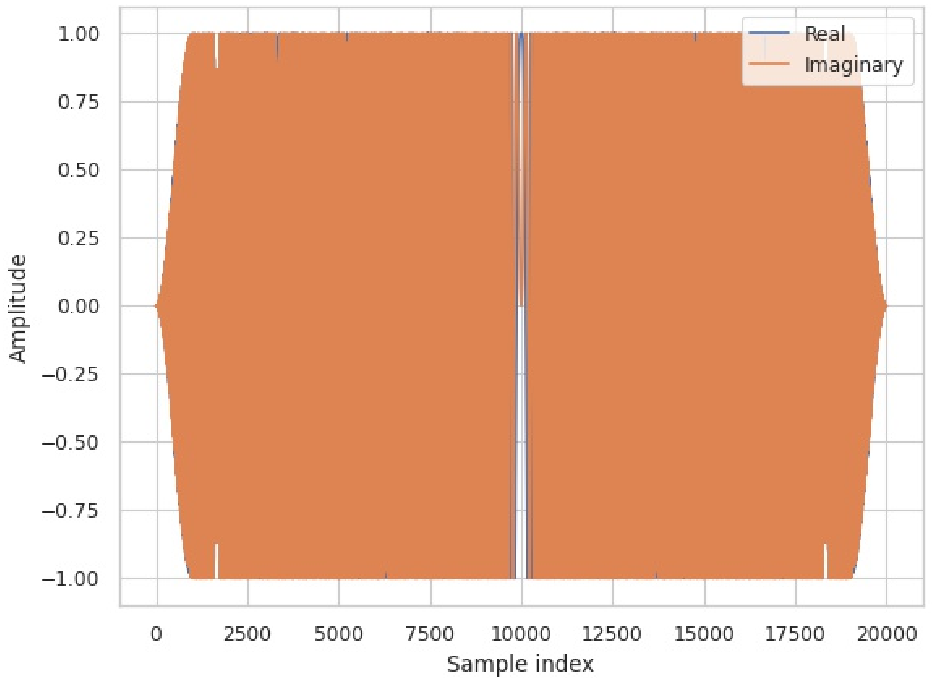


Figure 5.3: The ideal windowed chirp used in the predistortion testbench as the goal waveform.

bandwidth, the Ettus X310 was set up to run at the full $200\,\mathrm{MHz}$ sample rate, and the center frequency of the radio was set to $435\,\mathrm{MHz}$. Due to the limitations of the analog signal chain, the radar waveform used was a windowed $120\,\mathrm{MHz}$ linear frequency modulated chirp, shown in Figure 5.3. The window used was a Tukey window with 10% of the waveform in the cosine tapered region, so as to control the rise- and fall-time of the chirp. With a waveform to transmit, the testbench then creates a thread that continuously transmits the radar waveform spaced apart at some constant pulse repetition frequency using Python's `concurrent.futures` module.

Once the transmit thread has been started, the testbench lets the transmitter warm up for a small period of time to allow the transmitter to reach a steady state. The testbench then records the loopback signal received by the SDR for several pulse repetition periods. With knowledge of the pulse input to the transmitter and observations of the transmitted signal, the testbench then extracts the transmitted pulses from the larger signal. This was done by applying a matched filter to the received waveforms and using the largest peaks to determine pulse locations.

Once the received pulses have been extracted, one of the algorithms from Chapter 3 is applied to design a predistorted waveform. The system then starts transmitting the new, predistorted waveform, and this cycle continues until the desired stopping point. Finally, after all the desired data has been collected the testbench goes through the saved waveforms and estimates the values of the figures of merit of interest.

Using this testbench, many measurements of the performance of the algorithms discussed in Chapter 3 were gathered. The next chapter will present the results of those measurements.

Chapter 6

Results

Using the software defined radio testbench discussed in Chapter 5, the optimal Chebyshev filter predistortion algorithm and frequency windowing predistortion algorithm discussed in Chapter 3 were implemented, and their performance across multiple iterations was measured. This chapter begins by looking at the effect of the predistortion algorithms on the waveforms in the time domain before moving on to quantifying the impact of the predistortion algorithms on the range resolution and sidelobe level of the transmitted chirps. Finally, the spectral content of the transmitted chirps is briefly investigated.

6.1 Time Domain Measurements

Figure 6.1 shows the envelope magnitude of a transmitted pulse when the ideal chirp of Figure 5.3 is used as the input to the transmitter. It can be seen that there is significant variation in the pulse envelope throughout the duration of the pulse.

Using this waveform, the optimal Chebyshev filter predistortion algorithm discussed in Chapter 3 was applied, resulting in an FIR filter with the impulse response shown in Figure 6.2 and the frequency response shown in Figure 6.3.

Applying this correction to the ideal windowed chirp, the pulse output from the

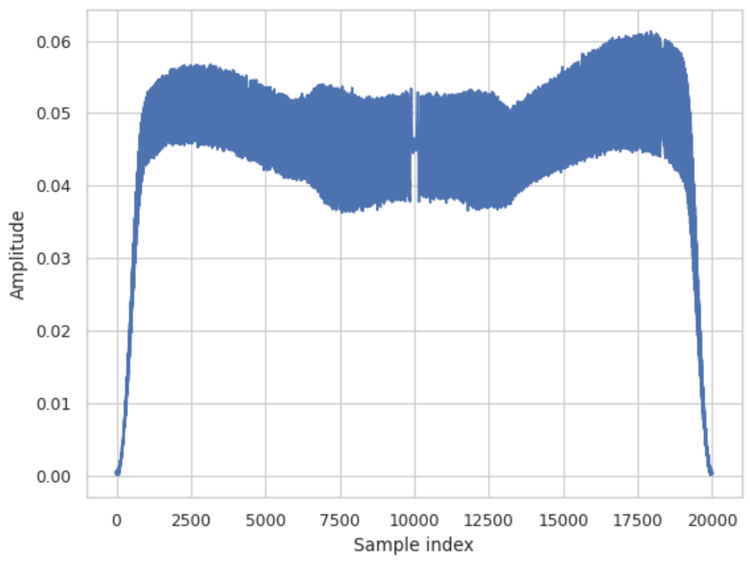


Figure 6.1: Magnitude of a transmitted chirp, seen before applying the optimal Chebyshev predistortion algorithm.

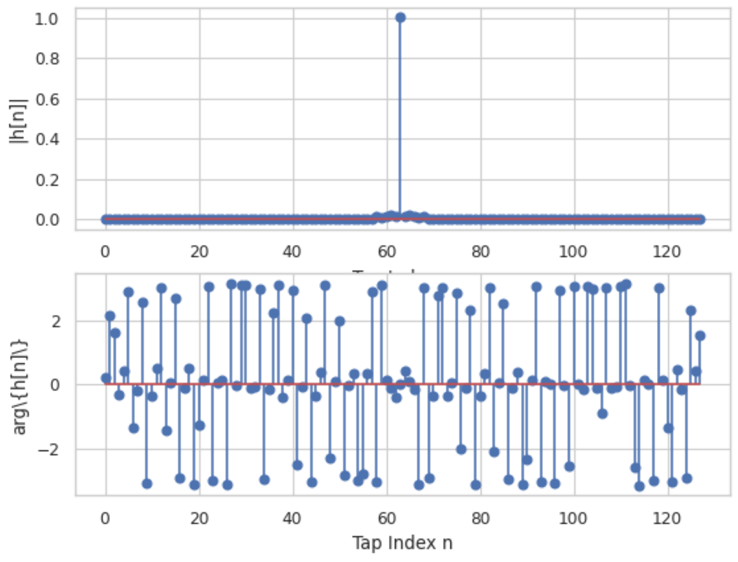


Figure 6.2: FIR filter taps obtained after the first iteration of the optimal Chebyshev predistortion algorithm.

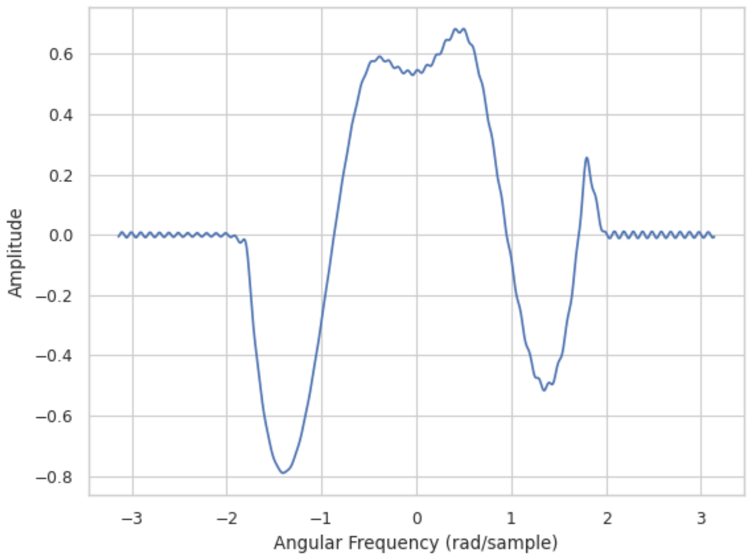


Figure 6.3: FIR filter magnitude frequency response obtained after the first iteration of the optimal Chebyshev predistortion algorithm.

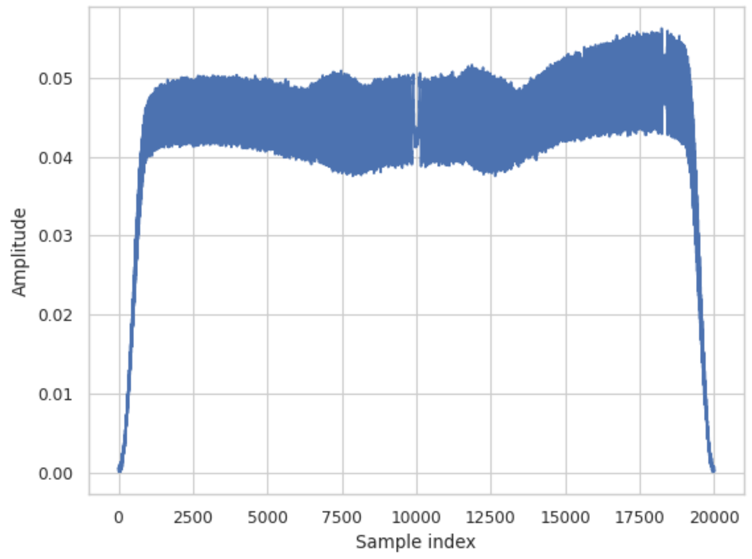


Figure 6.4: Magnitude of the transmitted chirp obtained after the first iteration of the optimal Chebyshev predistortion algorithm.

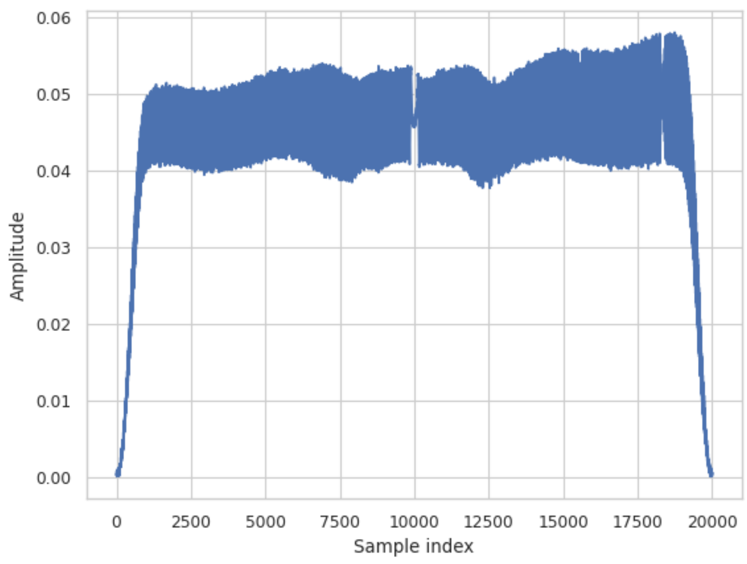


Figure 6.5: Magnitude of a transmitted chirp obtained after the ninth iteration of the optimal Chebyshev predistortion algorithm.

transmitter was that shown in Figure 6.4. It can be seen that there is still some ripple in the envelope of the waveform, but it has been reduced compared to Figure 6.1. Additionally, the pulse amplitude is slightly lower than in Figure 6.1, which is expected of most predistortion algorithms unless amplifier biasing is changed along with the application of predistortion. As the predistortion algorithm goes through successive iterations, the envelope magnitude becomes smoother, as can be seen in Figure 6.5.

Figure 6.6 again shows the transmitter output corresponding to an uncorrected pulse input. This pulse, however, was used as an input to the windowing predistortion algorithm discussed in Chapter 3. The estimated transfer function, along with the associated corrective window is shown in Figure 6.7, where the blue line is the estimated system transfer function before smoothing, the orange line is the estimated transfer function after smoothing, and the green line is the corrective window

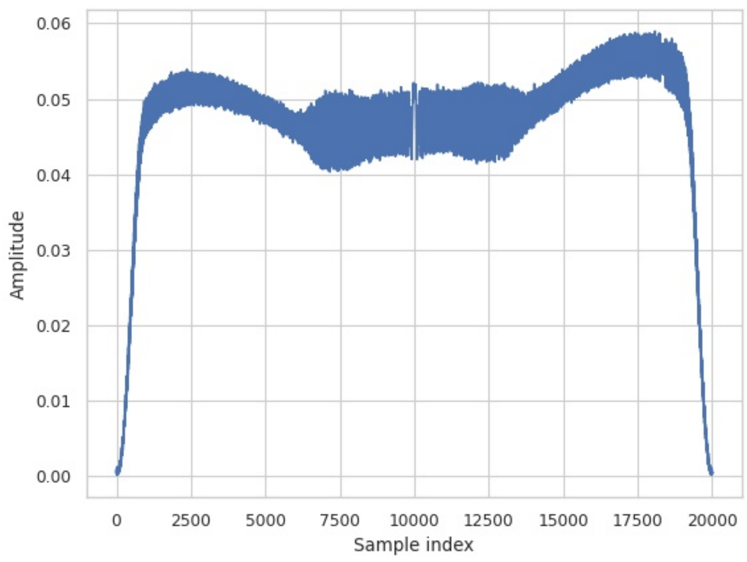


Figure 6.6: Magnitude of a transmitted chirp, seen before applying the windowing predistortion algorithm.

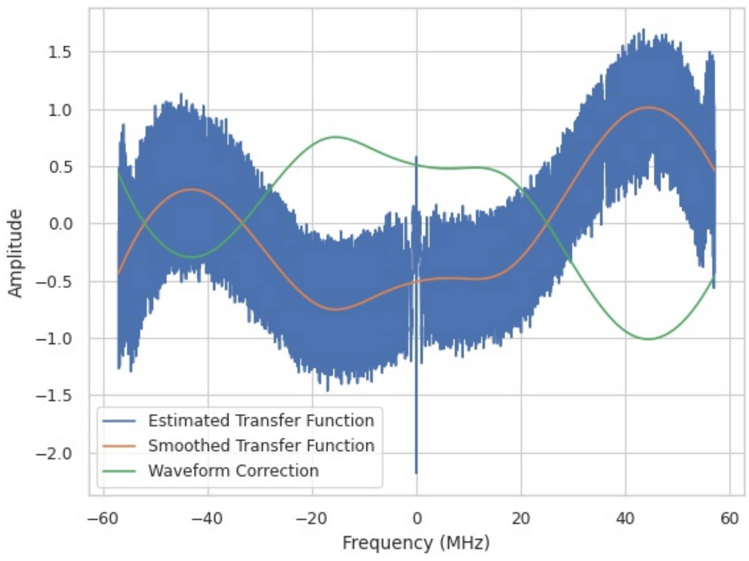


Figure 6.7: Estimated frequency response, and corresponding corrective window designed at the first iteration of the windowing predistortion algorithm.

function.

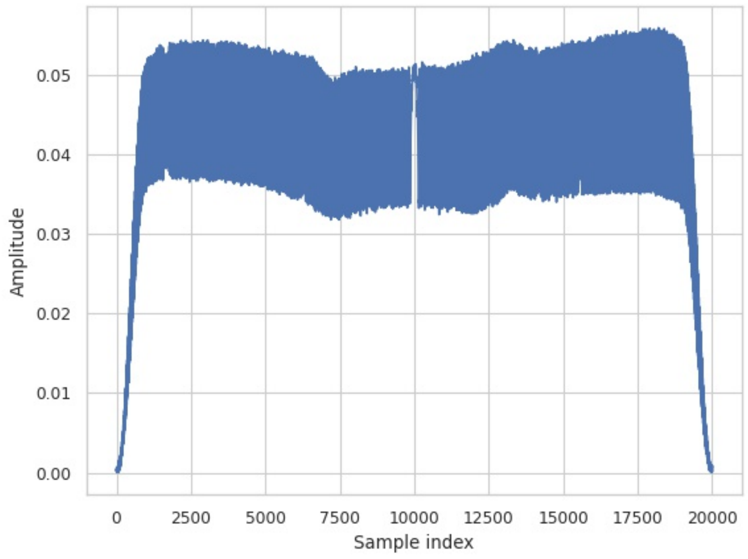


Figure 6.8: Magnitude of a transmitted chirp seen after the first iteration of the windowing predistortion algorithm.

After applying the corrective window from Figure 6.7, the observed output from the transmitter was that shown in Figure 6.8. Empirically, it appears that the pulse after this first iteration has a somewhat flatter envelope amplitude than after the first iteration of the Chebyshev filter algorithm. As before, successive iterations of the algorithm result in further improvements, as can be seen in Figure 6.9.

6.2 Impact on Range Resolution and Sidelobe Level

Having seen the effect of the predistortion algorithms discussed in Chapter 3 on time-domain waveforms, the discussion turns towards attempting to quantify how the radar waveforms may be improved. As was mentioned in Chapter 2, the main metrics to be used to judge the performance of the predistortion algorithms herein are the range resolution, effective bandwidth, integrated sidelobe ratio, and peak

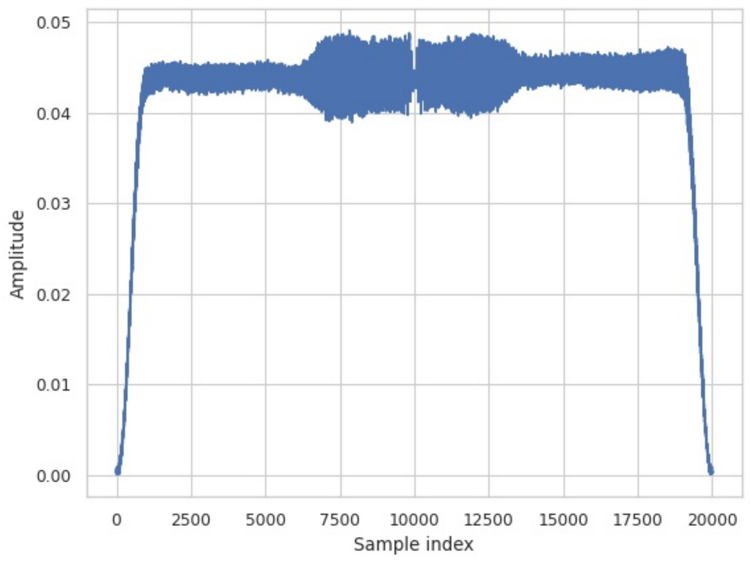


Figure 6.9: Magnitude of a transmitted chirp seen after the ninth iteration of the windowing predistortion algorithm.

sidelobe level of the transmitted waveform.

Measurements of these quantities from the SDR testbench running the optimal Chebyshev filter predistortion algorithm over 100 iterations are shown in Figures 6.10 to 6.14. In the plots presented here as a time series, specifically Figures 6.10 to 6.13, the first point (iteration 0) corresponds to the transmitted waveform when the ideal windowed chirp was used as the input to the transmitter, and the following points correspond to successive applications of the predistortion algorithm.

When applying pulse compression to the received waveforms to calculate the range resolution, peak sidelobe level, and integrated sidelobe ratio, there are two obvious waveforms that could be used to form the matched filter. First, the ideal chirp waveform could be used, which has the benefit of not containing noise and being relatively easily to apply in real-time inside an FPGA. Alternatively, the extracted pulse could be used, which has the advantage of being closer to the true matched filter (that is, the transmitted signal). Unfortunately, this waveform is embedded in some level of noise, and this calculation is relatively difficult to perform in real time on an FPGA. As such, pulse performance metrics will first be presented using the ideal waveform to form the matched filter, followed by measurements obtained using the extracted pulses for pulse compression.

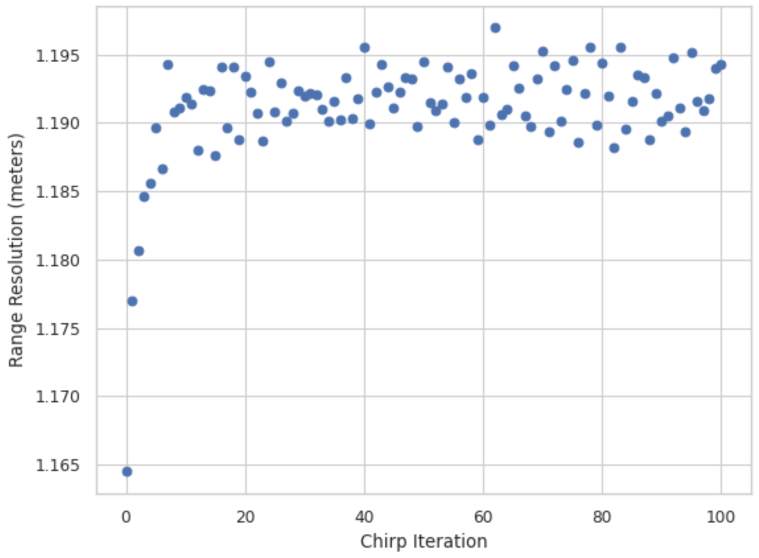


Figure 6.10: Observed range resolution across iterations of the FIR filter based predistortion algorithm.

Figure 6.10 shows the observed range resolution of the transmitted pulses across iterations of the optimal Chebyshev filter predistortion algorithm. While there are several ways to characterize the range resolution of a radar waveform in a pulse-Doppler system, it is defined here to be the $3\,\mathrm{dB}$ width of the pulse compressed transmit waveform, with the matched filter being derived from the ideal windowed chirp. It can be seen here that the range resolution seems to be degraded as a result of applying the predistortion algorithm. This is not, however, an entirely unexpected result as the expected range resolution of an ideal $120\,\mathrm{MHz}$ LFM chirp with a rectangular window is

$$\Delta R = \frac{c}{2B} \approx 1.25\,\mathrm{m}.$$

As a result, one interpretation of Figure 6.10 is that the range resolution is converging to the theoretical value as a result of the application of the predistortion algorithm. Convergence is used in this case, and throughout the remainder of this chapter, to mean the observations may have some random fluctuations, but the apparent mean value of the observations has reached an approximately constant value.

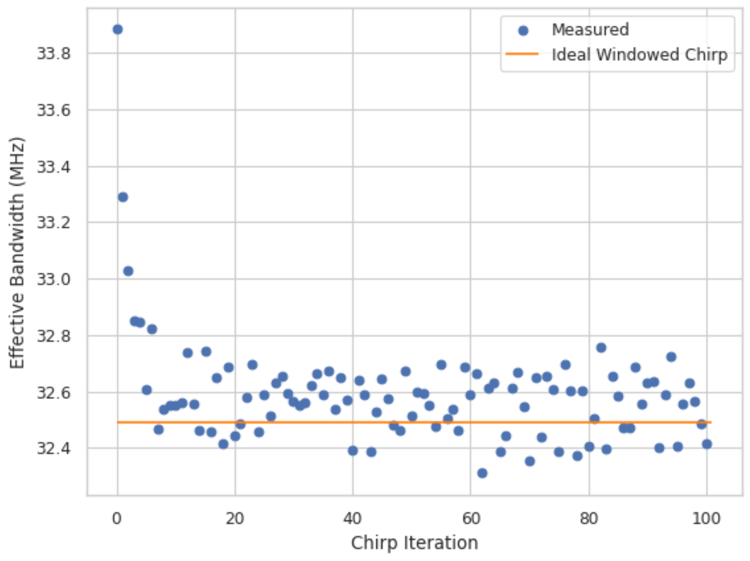


Figure 6.11: Observed effective bandwidth across iterations of the FIR filter based predistortion algorithm.

The measured effective bandwidth of the transmitted pulses across iterations of the optimal Chebyshev filter predistortion algorithm is shown in Figure 6.11. It can be seen here that the effective bandwidth quickly converges to be near the ideal windowed chirp's theoretical effective bandwidth of $32.49\,\mathrm{MHz}$. These numbers may seem low at first for a waveform that spans $120\,\mathrm{MHz}$ of spectrum, however they are quite reasonable when one considers that the effective bandwidth is the root-mean-square deviation of the signal's frequency from its center frequency, or in other words, the signal's standard deviation from the center frequency. LFM chirps have their power evenly distributed across their bandwidth, and the standard deviation of a uniform distribution from $-60\,\mathrm{MHz}$ to $60\,\mathrm{MHz}$ happens to be

$$\sigma = \sqrt{\frac{(120\,\mathrm{MHz})^2}{12}} \approx 34.64\,\mathrm{MHz}.$$

Since the windowed chirp is attenuated at the extremes of its frequency spectrum, it is expected that the effective bandwidth of the windowed chirp will be slightly less than an LFM chirp with no window applied, and the results of Figure 6.11 appear quite reasonable.

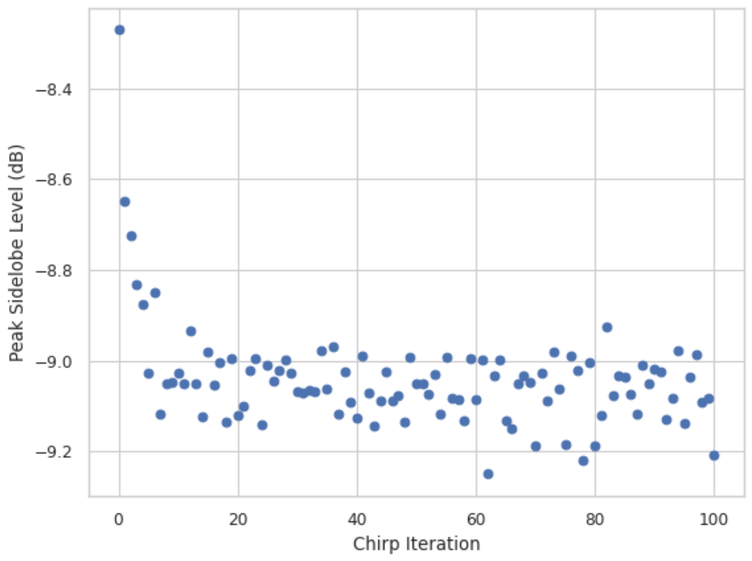


Figure 6.12: Observed peak sidelobe level across iterations of the FIR filter based predistortion algorithm.

Next, the peak sidelobe level of the transmitted pulses is shown across successive iterations of the optimal Chebyshev filter predistortion algorithm in Figure 6.12. A notable improvement in the peak sidelobe level can be seen as a result of application of the predistortion algorithm, with the peak sidelobe level going down by about $0.7\,\mathrm{dB}$ after several iterations of the algorithm. As with the range resolution and effective bandwidth, the improvement in peak sidelobe level seems to reach a sort of asymptote after about 20 iterations.

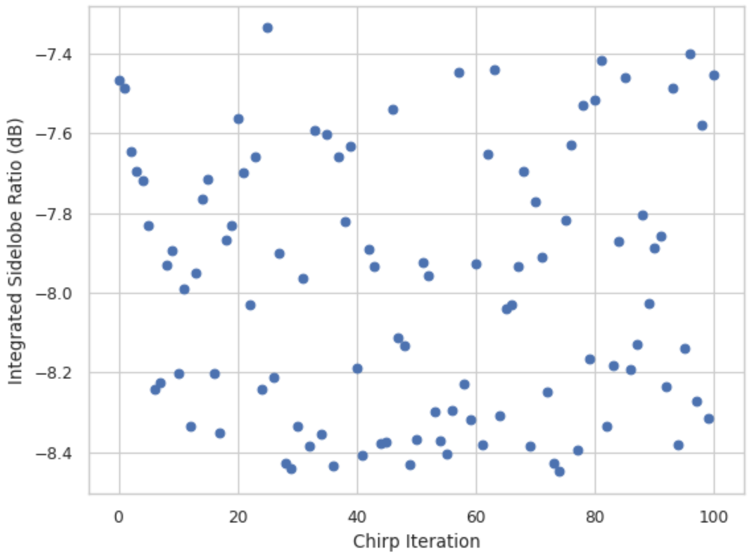


Figure 6.13: Observed integrated sidelobe ratio across iterations of the FIR filter based predistortion algorithm.

While the peak sidelobe level is of interest in judging the performance of radar waveforms, another important figure of merit is the integrated sidelobe ratio, which is shown for the optimal Chebyshev filter predistortion algorithm across iterations of the waveform in Figure 6.13. In contrast to Figures 6.10 to 6.12, the ISLR does not appear to have any correlation to how many iterations the predistortion algorithm has gone through. Indeed, it is difficult to claim any improvement in the ISLR. One interpretation of this fact is that while some power may be moved from the peak sidelobes to the main lobe, a large portion of the improvement in the peak sidelobe level might be a result of power being redistributed from the primary sidelobes to smaller sidelobes. While this is not necessarily a bad thing, it is certainly an interesting feature to note.

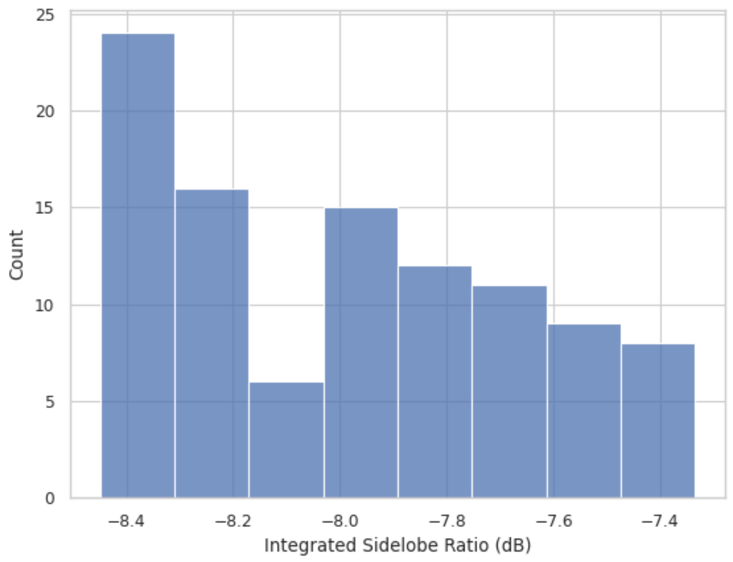


Figure 6.14: Observed distribution of ISLR measurements with the FIR filter based predistortion algorithm.

To look for other trends in the integrated sidelobe ratio, a histogram of the ISLR measurements from the optimal Chebyshev filter predistortion algorithm was created, shown in Figure 6.14. This histogram appears to show the ISLR measurements concentrated towards the lower end of the range of observed ISLR values. It might then be hypothesized that the optimal Chebyshev filter predistortion algorithm results in lower ISLR values on average, but will somewhat randomly generate waveforms that have a relatively large ISLR. However, it is likely not possible to draw this conclusion based on these observations alone, and this work makes no claim that there is a theoretical framework that might satisfyingly explain these observations, as such an endeavor is beyond the scope of this work.

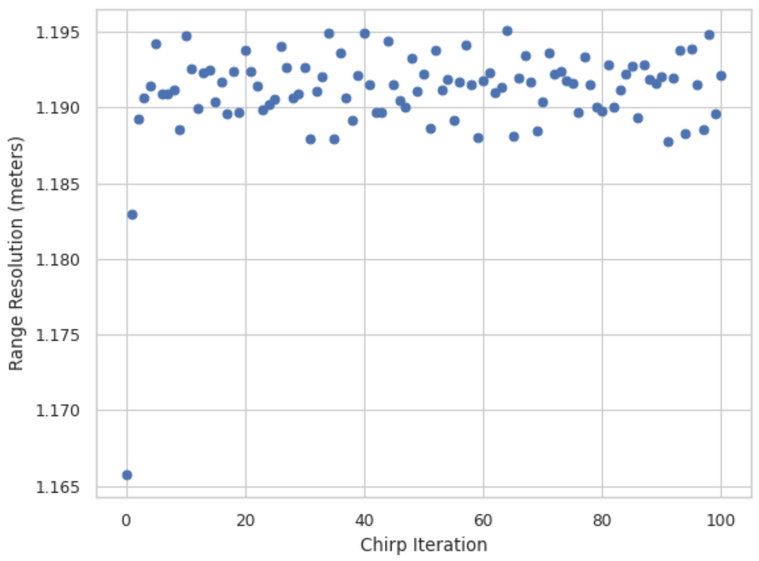


Figure 6.15: Observed range resolution across iterations of the amplitude correction predistortion algorithm.

Similar to Figure 6.10, Figure 6.15 shows the observed range resolution of the transmitted pulses across iterations of the windowing predistortion algorithm. As with the optimal Chebyshev filter predistortion algorithm, the range resolution initially experiences some degradation as a result of the predistortion algorithm, but quickly reaches a sort of steady-state asymptote. As with the Chebyshev filter algorithm, the range resolution gets closer to, but does not degrade to become wider than the theoretical resolution of the ideal chirp waveform. Compared to the FIR approach, it appears that the windowing approach has a similar steady-state range of range resolution observations. As before, iteration 0 corresponds to the observation of the non-predistorted chirp waveform in the time series plots of Figures 6.15 to 6.18.

Figure 6.16 shows the observed effective bandwidth across iterations of the am-

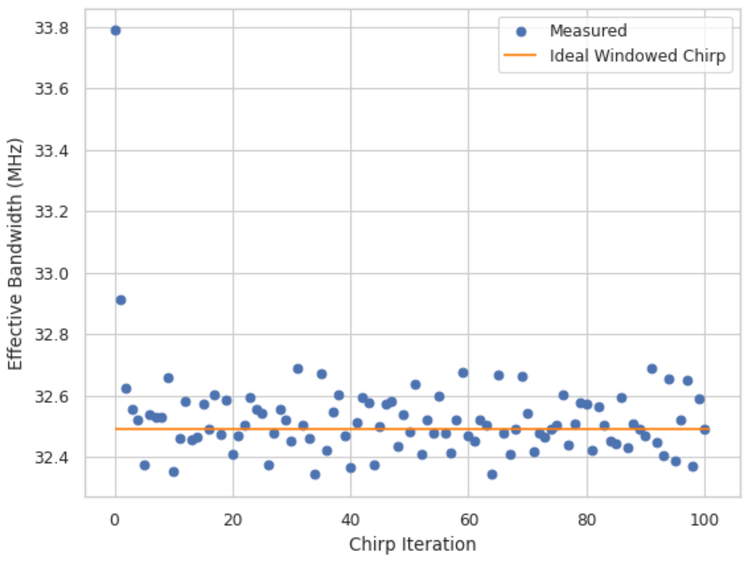


Figure 6.16: Observed effective bandwidth across iterations of the amplitude correction predistortion algorithm.

plitude correction predistortion algorithm. Similar to the optimal Chebyshev filter predistortion algorithm, the effective bandwidth quickly converges to the theoretical effective bandwidth of the ideal windowed chirp. Comparing Figures 6.11 and 6.16, it appears that the amplitude correction algorithm converges to the theoretical effective bandwidth somewhat quicker than the optimal Chebyshev filter design algorithm.

Figure 6.17 is the compliment to Figure 6.12, showing the peak sidelobe level of the observed transmitted waveform as a function of waveform iterations of the amplitude correction predistortion algorithm. As with the optimal Chebyshev filter predistortion algorithm we see the peak sidelobe level quickly decrease before it reaches a steady state where there is some fluctuation in the peak sidelobe level, but the observations remain within a relatively narrow range of values. Compared to the optimal FIR approach, the amplitude correction algorithm once again seems to

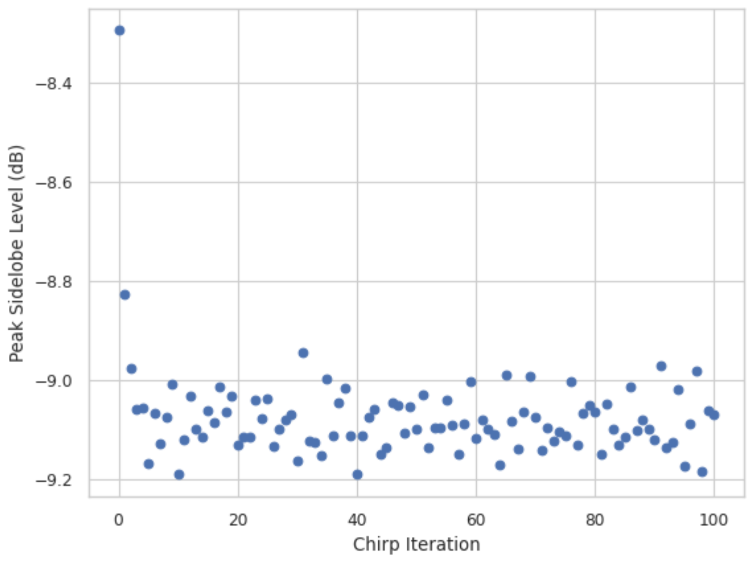


Figure 6.17: Observed peak sidelobe level across iterations of the amplitude correction predistortion algorithm.

converge somewhat faster.

The integrated sidelobe level of the amplitude correction predistortion algorithm is plotted as a function of waveform iteration in Figure 6.18. Similar to the optimal Chebyshev filter design algorithm, there does not appear to be much correlation between the observed integrated sidelobe ratio and the running of the predistortion algorithm. As was done earlier, a histogram of the observed integrated sidelobe ratios was created, shown in Figure 6.19. The distribution observed here seems to show the observed ISLR being concentrated in the center of the range in contrast to what was seen in Figure 6.14. It should also be noted that the range of the ISLR measurements are similar for both algorithms, ranging from approximately $-7.5\,\mathrm{dB}$ to $-8.5\,\mathrm{dB}$.

In this chapter, many measurements of the algorithms discussed in Chapter 3 were presented. The changes in the waveforms over time were observed, and it

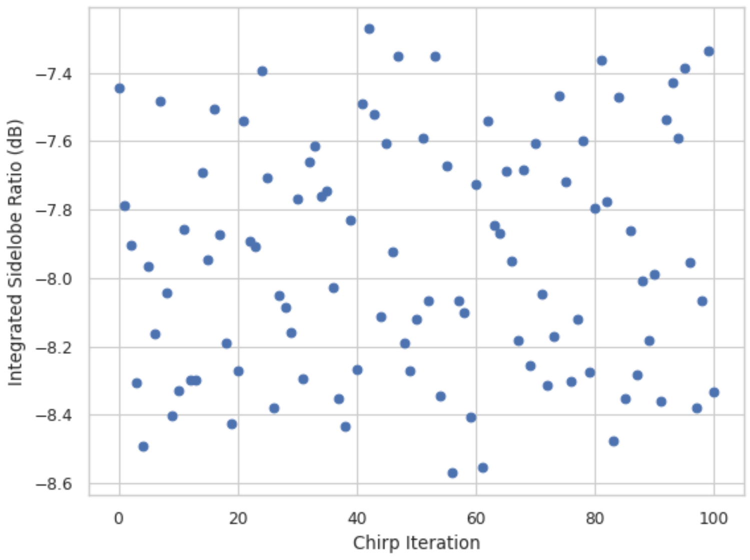


Figure 6.18: Observed integrated sidelobe ratio across iterations of the amplitude correction predistortion algorithm.

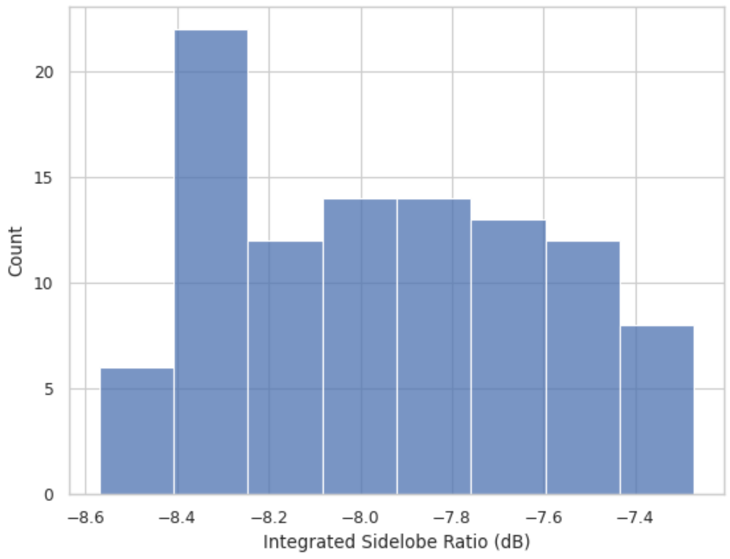


Figure 6.19: Observed distribution of ISLR measurements with the amplitude correction predistortion algorithm.

was seen that the envelope amplitude was smoothed as a result of predistortion. Additionally, the impact of predistortion on several important radar waveform performance metrics was observed. Having discussed these results, the next chapter will conclude this work.

6.2.1 Pulse Compression Using Extracted Pulses

This section presents measurements of pulse performance across iterations of the FIR filter and windowing predistortion algorithms, with the extracted pulses used to form the matched filter used for pulse compression rather than the ideal windowed chirp. It will be seen that this pulse compression approach causes the range resolution and peak sidelobe level to trend towards theoretical levels. Effective bandwidth is not revisited in this section, as the calculation of effective bandwidth does not use the pulse compressed waveform. Keeping with the convention so far, iteration 0 in Figures 6.20 to 6.22 and Figures 6.24 to 6.26 corresponds to the observation of a pulse input to the transmitter with no correction applied.

Figure 6.20 shows the observed range resolution across iterations of the FIR filter based predistortion algorithm using extracted pulses for matched filtering. The behavior observed in Figure 6.20 is similar to that seen in Figure 6.10, however the steady state mean in this case is near the theoretical mean of $1.178\,\mathrm{m}$.

Figure 6.21 shows the observed peak sidelobe level across iterations of the FIR filter based predistortion algorithm using extracted pulses for matched filtering. As with the range resolution, the overall behavior of the graph resembles that of the other pulse compression approach, however once again the steady state mean is near the theoretical level of $-13.29\,\mathrm{dB}$.

Figures 6.22 and 6.23 show the observed integrated sidelobe ratio across iter-

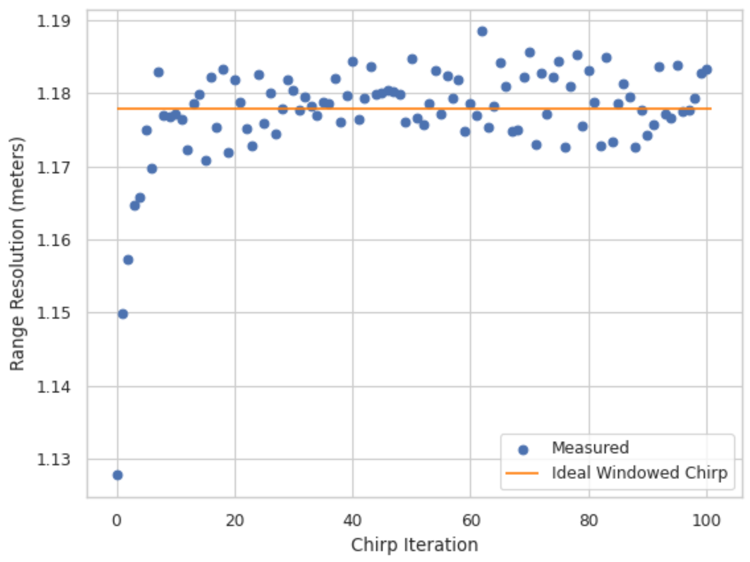


Figure 6.20: Observed range resolution across iterations of the FIR filter based predistortion algorithm using extracted pulses for matched filtering.

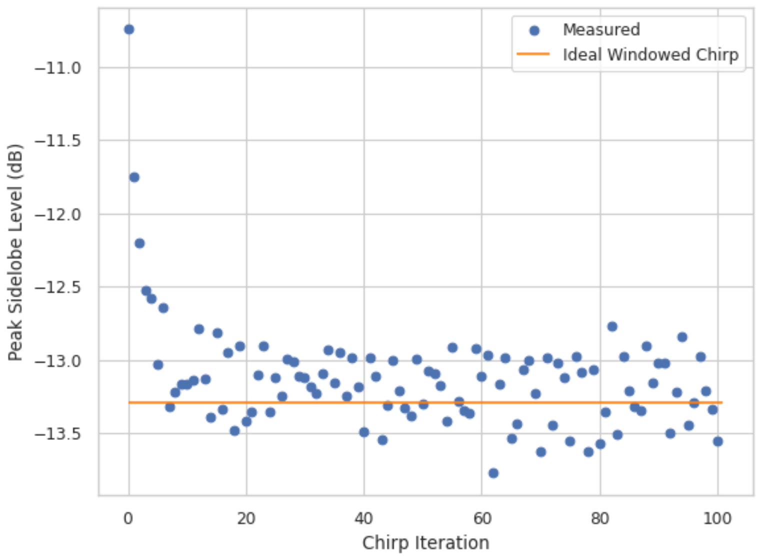


Figure 6.21: Observed peak sidelobe level across iterations of the FIR filter based predistortion algorithm using extracted pulses for matched filtering.

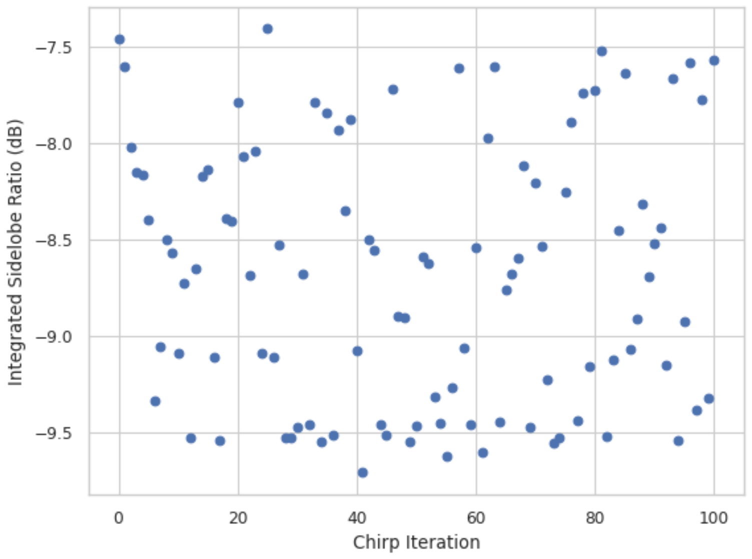


Figure 6.22: Observed integrated sidelobe ratio across iterations of the FIR filter based predistortion algorithm using extracted pulses for matched filtering.

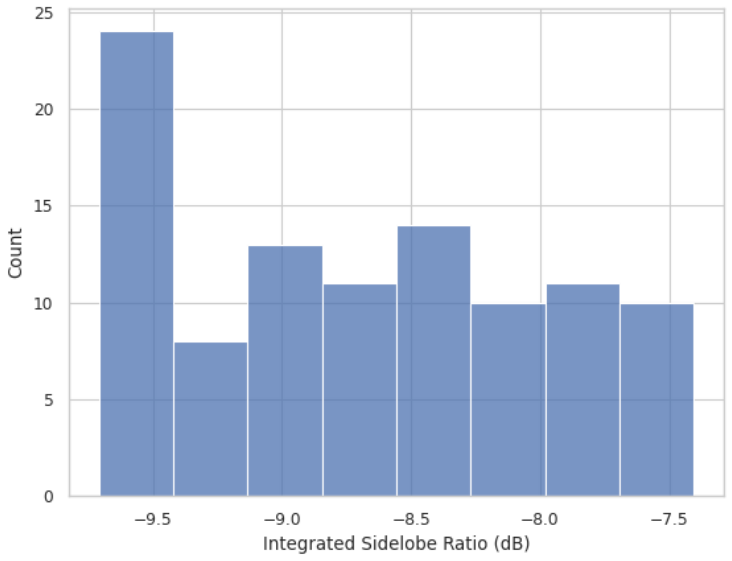


Figure 6.23: Observed distribution of integrated sidelobe ratio measurements with the FIR filter based predistortion algorithm using extracted pulses for matched filtering.

ations of the FIR filter based predistortion algorithm and the corresponding distribution of ISLR measurements, respectively. As was seen with the measurements using the ideal chirp as the matched filter, there is no notable correlation between iterations of the predistortion algorithm and the observed integrated sidelobe ratio.

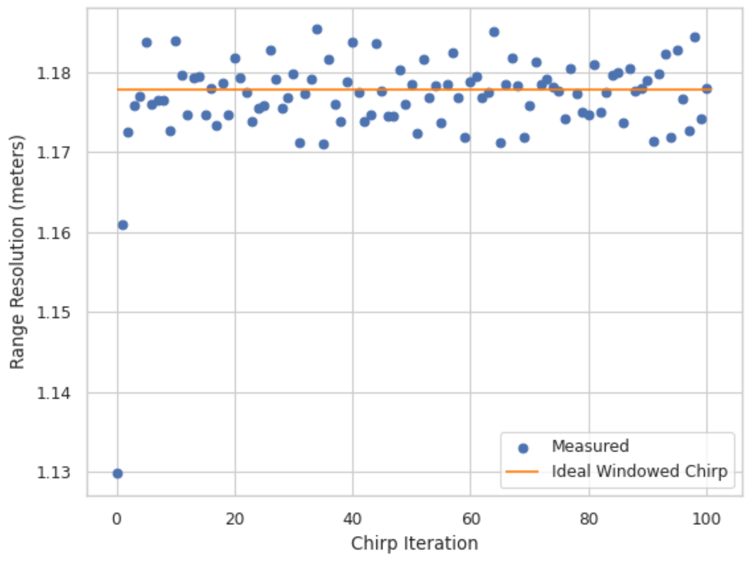


Figure 6.24: Observed range resolution across iterations of the windowing predistortion algorithm using extracted pulses for matched filtering.

Figure 6.24 shows the observed range resolution across iterations of the windowing predistortion algorithm using extracted pulses for matched filtering. As was seen when using the ideal chirp in the pulse compression process, the windowing algorithm reaches its steady state faster than the FIR filter algorithm. Once again, the use of extracted pulses to apply pulse compression results in a steady state mean near the theoretical range resolution of $1.178\,\mathrm{m}$.

Figure 6.25 shows the observed peak sidelobe level across iterations of the windowing predistortion algorithm using extracted pulses for the matched filtering process. Compared to Figure 6.17, the most notable difference is that the observed

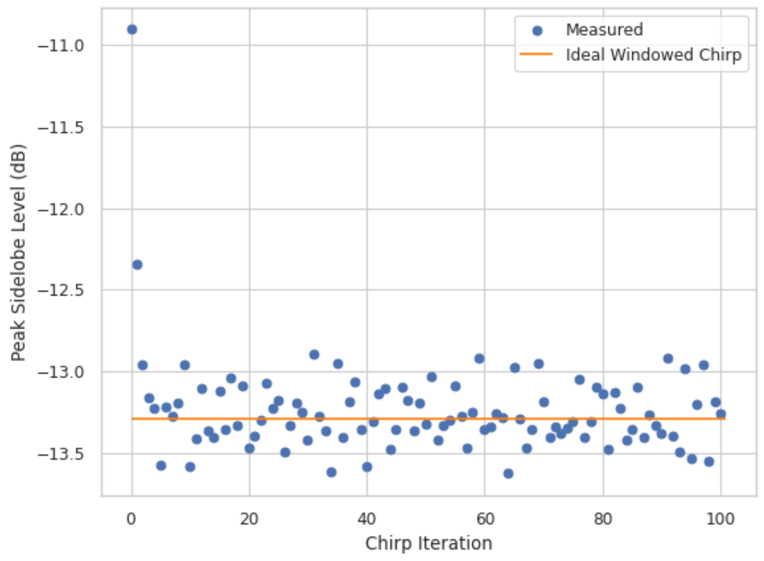


Figure 6.25: Observed peak sidelobe level across iterations of the windowing predistortion algorithm using extracted pulses for matched filtering.

peak sidelobe level has a steady state mean near the theoretical value of $-13.29\,\mathrm{dB}$.

Figures 6.26 and 6.27 show the observed integrated sidelobe ratio across iterations of the windowing predistortion algorithm and the distribution of these ISLR measurements respectively. As was seen in Figures 6.18 and 6.19, there is no notable correlation between the observed integrated sidelobe ratio and iterations of the windowing predistortion algorithm.

To enable better comparison of the performance of the two predistortion algorithms, the approximate value of some interesting features of the above pulse performance graphs were estimated and summarized in Table 6.1.

6.3 Frequency Domain Performance

To observe the effect of the FIR filter based predistortion algorithm on the spectral content of the transmitted waveforms, the normalized power spectral density

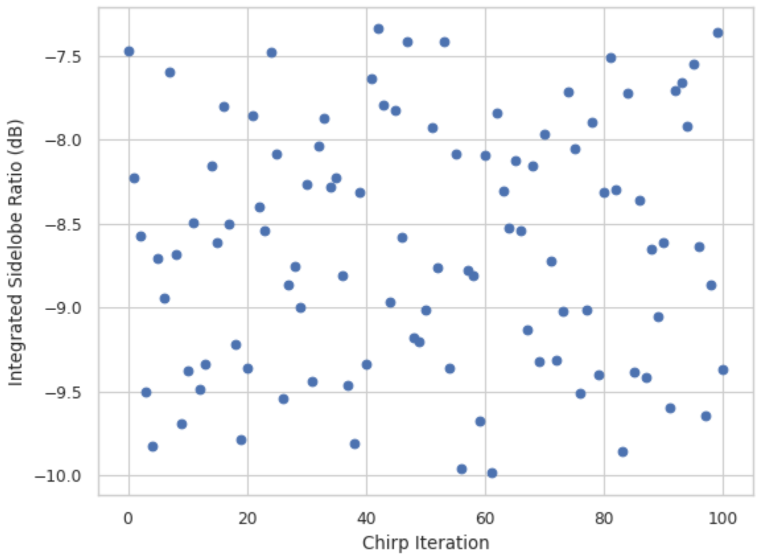


Figure 6.26: Observed integrated sidelobe ratio across iterations of the windowing predistortion algorithm using extracted pulses for matched filtering.

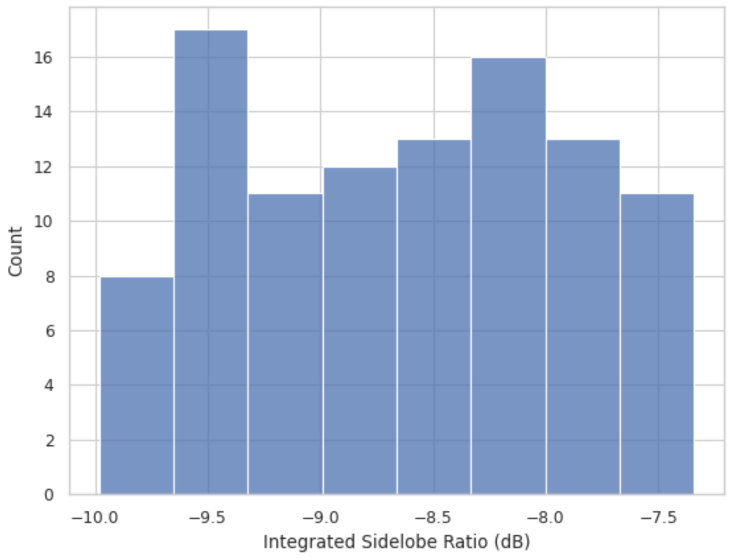


Figure 6.27: Observed distribution of integrated sidelobe ratio measurements with the windowing predistortion algorithm using extracted pulses for matched filtering.

FIR Filter Algorithm, Ideal Pulse Matched Filter			
Measurement	Starting value	Iterations Before Steady State	Steady State Mean
Range Resolution (m)	1.165	8	1.193
Effective Bandwidth (MHz)	33.9	8	32.5
Peak Sidelobe Level (dB)	-8.25	8	-9.1

Windowing Algorithm, Ideal Pulse Matched Filter			
Measurement	Starting value	Iterations Before Steady State	Steady State Mean
Range Resolution (m)	1.165	3	1.193
Effective Bandwidth (MHz)	33.8	3	32.5
Peak Sidelobe Level (dB)	-8.3	3	-9.1

FIR Filter Algorithm, Extracted Pulse Matched Filter			
Measurement	Starting value	Iterations Before Steady State	Steady State Mean
Range Resolution (m)	1.127	7	1.18
Effective Bandwidth (MHz)	33.9	8	32.5
Peak Sidelobe Level (dB)	-10.8	8	-13.3

Windowing Algorithm, Extracted Pulse Matched Filter			
Measurement	Starting value	Iterations Before Steady State	Steady State Mean
Range Resolution (m)	1.13	3	1.178
Effective Bandwidth (MHz)	33.8	3	32.5
Peak Sidelobe Level (dB)	-10.9	3	-13.3

Table 6.1: Comparison of the approximate values of some important features of the graphs of the pulse performance metrics that displayed convergence to a steady state.

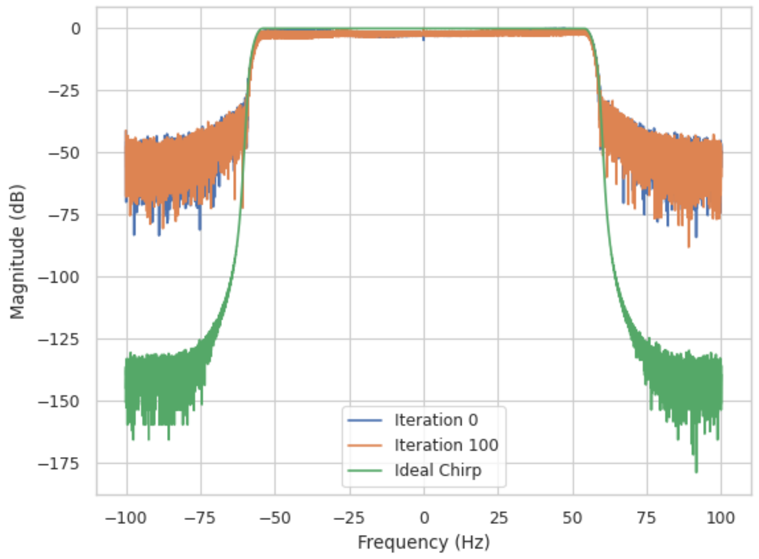


Figure 6.28: Normalized estimated power spectral density of the transmitted chirps from the FIR filter algorithm observations, along with the spectrum of the ideal windowed chirp.

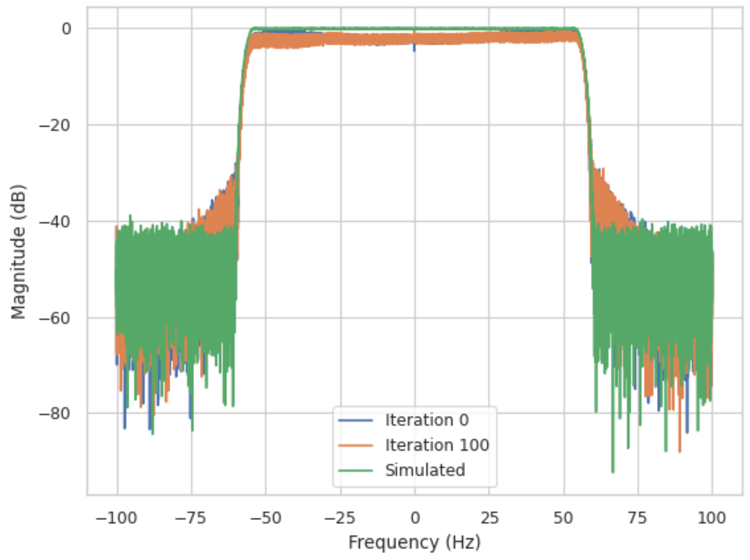


Figure 6.29: Normalized estimated power spectral density of the transmitted chirps from the FIR filter algorithm observations, along with the estimated spectrum of the ideal chirp combined with simulated white noise.

of the transmitted waveforms before any predistortion and after 100 iterations of the algorithm was estimated, and is shown in Figure 6.28. It can be seen that the out-of-band spectral content is similar for both measured waveforms, and is over $75\,\mathrm{dB}$ higher than that calculated for an ideal windowed chirp. This comparison does not, however, account for the impact of additive white noise in the system. As such, the noise level was estimated using a testbench recording with nothing being transmitted, and simulated noise waveforms were created and added to the ideal waveform. The result is compared to the measured power spectral density in Figure 6.29, from which it can be seen that much of the discrepancy between the ideal spectrum and measured spectrum is due to the impact of white noise. Figure 6.29 still shows about $10\,\mathrm{dB}$ of spectral regrowth near the chirp spectrum above the noise floor, which does not seem to be significantly impacted by the predistortion algorithm. This is the expected result, as the predistortion algorithms were not designed to directly compensate for spectral regrowth due to the chirps occupying most of the data converter bandwidth.

Similarly, the normalized estimated power spectral density of the transmitted waveform before any chirp correction and after the 100th iteration of the windowing predistortion algorithm is shown in Figure 6.30 along with the spectrum of the ideal waveform combined with simulated noise. As with the FIR filter based algorithm, it can be seen that much of the out-of-band spectral content is accounted for by the noise present in the system, but there is about $10\,\mathrm{dB}$ of spectral regrowth in the transmitted waveforms above the noise floor that is apparently unaffected by the windowing algorithm. As with the FIR filter approach, this is expected due to the design of the algorithm.

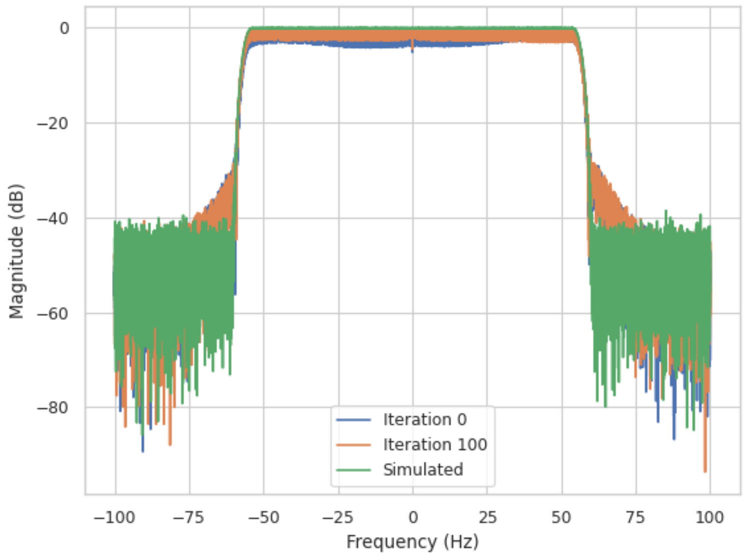


Figure 6.30: Normalized estimated power spectral density of the transmitted chirps from the windowing algorithm observations, along with the estimated spectrum of the ideal chirp combined with simulated white noise.

Chapter 7

Conclusion and Future Work

This work began by analyzing the requirements for predistortion systems demanded by LRM pulse radars and looking at existing predistortion algorithms in the context of radar applications. The specialization of the memory polynomial model to better suit radar applications was discussed, along with a windowing method to further reduce computational complexity and potentially result in faster convergence. To reduce the dependence of parameter estimation on time alignment, a frequency domain approach to parameter estimation was explored.

Additionally, the high-bandwidth data streams seen in modern digital radar architectures and some challenges encountered as a result were discussed. This discussion is used as motivation for the development of a modular FPGA-based radar data acquisition testbench for the validation of real-time radar DSP algorithms. A proposed design for the implementation of such a system was discussed, including networking aspects and the hardware internal to the FPGA. This design has the potential to enable the rapid testing of digital radar signal processing algorithms with real data using commercial off-the-shelf components.

Using an SDR based testbench, the predistortion algorithms discussed in Chapter 3 tailored to LFM pulse radar systems were tested. It was seen that these algorithms resulted in reduced distortion of the transmitted waveform, and in exchange for a marginal decrease of effective bandwidth, the peak range sidelobe level was significantly reduced by both algorithms. Overall, the radar waveform performance trended towards theoretical levels in several metrics, enabling more predictable and precise radar measurements.

Some possible interesting future work might involve implementing the algorithms of Chapter 3 in an FPGA as discussed in Chapter 4. Additionally, it seems that the SDR used for algorithm verification in this work supports time-stamped data streams, potentially allowing sample synchronization, and therefore the testing of predistortion algorithms with stronger dependence on sample synchronization.

References

[1] G. P. Fettweis, “A 5G wireless communications vision,” *Microwave Journal*, vol. 55, no. 12, pp. 24–36, 2012.

[2] L. Dai, B. Wang, Y. Yuan, S. Han, C. I, and Z. Wang, “Non-orthogonal multiple access for 5G: solutions, challenges, opportunities, and future research trends,” *IEEE Communications Magazine*, vol. 53, no. 9, pp. 74–81, sep 2015.

[3] C. Desset, B. Debaillie, V. Giannini, A. Fehske, G. Auer, H. Holtkamp, W. Wajda, D. Sabella, F. Richter, M. J. Gonzalez, H. Klessig, I. Godor, M. Olsson, M. A. Imran, A. Ambrosy, and O. Blume, “Flexible power modeling of LTE base stations,” in *2012 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, apr 2012.

[4] A. Banerjee, R. Hezar, and L. Ding, “Efficiency improvement techniques for RF power amplifiers in deep submicron CMOS,” in *2015 IEEE Custom Integrated Circuits Conference (CICC)*, Sep. 2015, pp. 1–4.

[5] R. N. Braithwaite and F. Luo, “General principles and design overview of digital predistortion,” *Digital Processing for Front End in Wireless Communication and Broadcasting*, pp. 143–191, 2011.

[6] J. Kim and K. Konstantinou, “Digital predistortion of wideband signals based on power amplifier model with memory,” *Electronics Letters*, vol. 37, no. 23, p. 1417, 2001.

[7] L. Ding, G. Zhou, D. Morgan, Z. Ma, J. Kenney, J. Kim, and C. Giardina, “A robust digital baseband predistorter constructed using memory polynomials,” *IEEE Transactions on Communications*, vol. 52, no. 1, pp. 159–165, jan 2004.

[8] D. Morgan, Z. Ma, J. Kim, M. Zierdt, and J. Pastalan, “A generalized memory polynomial model for digital predistortion of RF power amplifiers,” *IEEE Transactions on Signal Processing*, vol. 54, no. 10, pp. 3852–3860, oct 2006.

[9] C. Fager, T. Eriksson, F. Barradas, K. Hausmair, T. Cunha, and J. C. Pedro, “Linearity and efficiency in 5G transmitters: New techniques for analyzing efficiency, linearity, and linearization in a 5G active antenna transmitter context,” *IEEE Microwave Magazine*, vol. 20, no. 5, pp. 35–49, may 2019.

[10] P. L. Gilabert, G. Montoro, D. Vegas, N. Ruiz, and J. A. Garcia, “Digital predistorters go multidimensional: DPD for concurrent multiband envelope tracking and outphasing power amplifiers,” *IEEE Microwave Magazine*, vol. 20, no. 5, pp. 50–61, may 2019.

[11] R. Summers, M. Yeary, H. Sigmarsson, and R. Rincon, “Adaptive Digital Predistortion for Radar Applications using Convex Optimization,” in *2020 IEEE International Radar Conference (RADAR)*, 2020, pp. 816–820.

[12] ——, “Architecture Study for a Bare-Metal Direct Conversion Radar FPGA Testbench,” in *2021 IEEE Radar Conference (RadarConf21)*, 2021, pp. 1–5.

[13] R. Rincon, L. Carter, D. Lu, C. D. Toit, M. Perrine, D. M. Hollibaugh-Baker, and C. D. Neish, “Space Exploration Synthetic Aperture Radar (SESAR),” in *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. Yokohama, Japan: IEEE, Jul. 2019, pp. 8320–8323.

[14] M. L. Perrine, R. Rincon, S. Van Nostrand, H. Nguyen, M. A. Moe, H. H. Sigmarsson, and M. B. Yeary, “Miniaturized P-band beamforming synthetic aperture radar transceiver,” in *2018 IEEE Radar Conference (RadarConf18)*. Oklahoma City, OK: IEEE, Apr. 2018, pp. 1533–1536.

[15] R. F. Rincon, T. Fatoyinbo, G. Sun, K. J. Ranson, M. Perrine, M. Deshapnde, and Q. Bonds, “The EcoSAR P-band synthetic aperture radar,” in *2011 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, Jul. 2011.

[16] T. Fatoyinbo, R. F. Rincon, G. Sun, and K. J. Ranson, “EcoSAR: A P-band digital beamforming polarimetric interferometric SAR instrument to measure ecosystem structure and biomass,” in *2011 IEEE International Geoscience and Remote Sensing Symposium*. IEEE, Jul. 2011.

[17] M. Brandolini *et al.*, “A 5 GS/s 150 mW 10 b SHA-less pipelined/SAR hybrid ADC for direct-sampling systems in 28 nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 12, pp. 2922–2934, 2015.

[18] T. Chalvatzis, E. Gagnon, M. Repeta, and S. P. Voinigescu, “A Low-Noise 40-GS/s Continuous-Time Bandpass $\Delta\Sigma$ ADC Centered at 2 GHz for Direct Sampling Receivers,” *IEEE Journal of Solid-State Circuits*, vol. 42, no. 5, pp. 1065–1075, May 2007.

[19] R. Gomez, “Theoretical comparison of direct-sampling versus heterodyne RF receivers,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 8, pp. 1276–1282, 2016.

[20] J. Fang, S. Thirunakkarasu, X. Yu, F. Silva-Rivas, C. Zhang, F. Singor, and J. Abraham, "A 5-GS/s 10-b 76-mW time-interleaved SAR ADC in 28 nm CMOS," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 64, no. 7, pp. 1673–1683, 2017.

[21] M. Guo, J. Mao, S.-W. Sin, H. Wei, and R. P. Martins, "A 5 GS/s 29 mW interleaved SAR ADC with 48.5 dB SNDR using digital-mixing background timing-skew calibration for direct sampling applications," *IEEE Access*, vol. 8, pp. 138 944–138 954, 2020.

[22] L. Kull, D. Luu, C. Menolfi, M. Braendli, P. A. Francese, T. Morf, M. Kossel, A. Cevrero, I. Ozkaya, and T. Toifl, "A 24–72-GS/s 8-b time-interleaved SAR ADC with 2.0–3.3-pJ/conversion and 30 dB SNDR at Nyquist in 14-nm CMOS FinFET," *IEEE Journal of Solid-State Circuits*, vol. 53, no. 12, pp. 3508–3516, 2018.

[23] M. Schetzen, *The Volterra and Wiener theories of nonlinear systems / Martin Schetzen.* New York: Wiley, 1980.

[24] A. Zhu, "Behavioral modeling for digital predistortion of RF power amplifiers: from Volterra series to CPWL functions," in *2016 IEEE Topical Conference on Power Amplifiers for Wireless and Radio Applications (PAWR)*, 2016, pp. 1–4.

[25] Z. Dunn, M. Yeary, C. Fulton, and N. Goodman, "Wideband digital predistortion of solid-state radar amplifiers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 52, no. 5, pp. 2452–2466, 2016.

[26] M. I. M. I. Skolnik, *Introduction to radar systems / Merrill I. Skolnik.*, 3rd ed. Boston: McGraw Hill, 2001.

[27] Z. Dunn, M. Yeary, C. Fulton, and N. Goodman, "Memory polynomial model for digital predistortion of broadband solid-state radar amplifiers," in *2015 IEEE Radar Conference (RadarCon).* IEEE, may 2015.

[28] S. Boyd and L. Vandenberghe, *Convex Optimization.* Cambridge University Press, Mar. 2004.

[29] M. S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret, "Applications of second-order cone programming," *Linear Algebra and its Applications*, vol. 284, no. 1-3, pp. 193–228, nov 1998.

[30] F. Alizadeh and D. Goldfarb, "Second-order cone programming," *Mathematical Programming*, vol. 95, no. 1, pp. 3–51, jan 2003.

[31] Texas Instruments, “DAC3484 data sheet, product information and support.” [Online]. Available: https://www.ti.com/product/DAC3484

[32] F. Azam, A. Karwankar, and G. Isola, “Generating timing and control signals for C-band T/R module using FPGA,” in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*. IEEE, 2018, pp. 1052–1055.

[33] M. B. Yeary, W. Zhang, O. Alkhouli, and K. Wong-Hagen, “Design of an FPGA-based RF link for data and power transfer,” *IEEE Transactions on Instrumentation and Measurement*, vol. 55, no. 6, pp. 2313–2319, 2006.

[34] R. F. Molanes, J. J. Rodríguez-Andina, and J. Farina, “Performance characterization and design guidelines for efficient processor–FPGA communication in Cyclone V FPSoCs,” *IEEE Transactions on Industrial Electronics*, vol. 65, no. 5, pp. 4368–4377, 2017.

[35] R. F. Molanes, L. Costas, J. J. Rodriguez-Andina, and J. Farina, “Comparative analysis of processor-FPGA communication performance in low-cost FPSoCs,” *IEEE Transactions on Industrial Informatics*, 2020.

[36] M. Petri and M. Ehrig, “A SoC-based SDR platform for ultra-high data rate broadband communication, radar and localization systems,” in *2019 Wireless Days (WD)*. IEEE, 2019, pp. 1–4.

[37] M. Frigo and S. G. Johnson, “The design and implementation of FFTW3,” *Proceedings of the IEEE*, vol. 93, no. 2, p. 16, 2005.

[38] Xilinx, “AXI DMA v7.1 LogiCORE IP Product Guide,” 2019. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf

[39] ARM, “AMBA 4 AXI4-Stream Protocol Specification.” [Online]. Available: https://developer.arm.com/documentation/ihi0051/latest

[40] B. Perez, “Bperez77/xilinx_axidma.” [Online]. Available: https://github.com/bperez77/xilinx_axidma

[41] Texas Instruments, “ADS5402 data sheet, product information and support — TI.com.” [Online]. Available: https://www.ti.com/product/ADS5402

[42] Xilinx, “7 Series FPGAs Configurable Logic Block User Guide (UG474),” 2016. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf

[43] ——, "7 Series FPGAs Clocking Resources User Guide (UG472)," 2018. [Online]. Available: https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf

Appendix A

Testbench Code

This appendix contains interesting selections of the Python code used to implement the software defined radio based predistortion testbench.

The segment below shows how a pulse transmit thread was started:

```
# Create a function to transmit the radar waveform
def transmit_waveform(sdr, tx_stream, waveform,
   tx_stop_event):
   while not tx_stop_event.is_set():
      rc = sdr.writeStream(tx_stream, [waveform], len(
         waveform))
      if rc.ret != len(waveform):
         print('TX␣Error␣{}:␣{}'.format(rc.ret, errToStr(rc.
            ret)))

# Start transmitting
tx = concurrent.futures.ThreadPoolExecutor(max_workers=1)
tx_stop_event = threading.Event()
tx_task = tx.submit(transmit_waveform, sdr, tx_stream,
   waveform, tx_stop_event)
```

The segment below shows the loop used to gather recordings of the transmitted waveform:

```
for idx in range(recording_periods):
  try:
     while True:
        if idx < num_warmup_recordings:
           sr = sdr.readStream(rx_stream, [rx_buff],
              rx_sample_count)
           print(f"RX Period {idx} - Num Samples: {sr.ret},
              Flags: {sr.flags}, Timestamp {sr.timeNs}")
        else:
           rx_buff = rx_save_buff[:,(idx-
              num_warmup_recordings)]
```

```
            sr = sdr.readStream(rx_stream, [rx_buff],
               rx_sample_count)
            print(f"RX Period {idx} - Num Samples: {sr.ret},
               Flags: {sr.flags}, Timestamp {sr.timeNs}")
         if sr.ret == rx_sample_count:
            break
   except KeyboardInterrupt:
      tx_stop_event.set()
      tx_task.result(timeout=1.0)
      break
```

The segment below shows the process used to extract individual pulses from the larger waveform:

```
rx_pcmag = np.abs(signal.correlate(rx_waveform, chirp))

buff_size = round(pri_sample_count * 0.7)
max_locations = []
current_max = 0
current_max_location = 0
for idx in range(rx_pcmag.size):
   if rx_pcmag[idx] > current_max:
      current_max_location = idx
      current_max = rx_pcmag[idx]
   if (idx - current_max_location) > buff_size:
      max_locations.append(current_max_location)
      current_max = 0

pc_maxima = rx_pcmag[max_locations]
pc_smaller_maxima = pc_maxima
pc_biggest_maxima_indices = []
for idx in range(num_pulses_to_use):
   biggest_idx = np.where(rx_pcmag == max(pc_smaller_maxima)
      )
   pc_biggest_maxima_indices.append(biggest_idx[0][0])
   pc_smaller_maxima = np.delete(pc_smaller_maxima, np.where
      (pc_smaller_maxima == rx_pcmag[biggest_idx]))

max_locations -= chirp_sample_count
pc_biggest_maxima_indices -= chirp_sample_count

for idx in range(num_pulses_to_use):
   start_idx = pc_biggest_maxima_indices[idx]
   end_idx = start_idx + chirp_sample_count
   chirp = rx_waveform[start_idx:end_idx]
   np.save(f"data/rx-chirps/{outprefix}chirp_{idx}.npy",
      chirp)
```

The segment below shows the code used to implement the optimal Chebyshev FIR filter predistortion algorithm:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy import interpolate
from scipy.fftpack import fft, fftshift, fftfreq
import argparse
from pathlib import Path
import cvxpy as cp
import seaborn as sns

sns.set_theme(context="paper", style="whitegrid")

def dbode(h, prefix, idx):
   w = np.linspace(-np.pi, np.pi, 1024)
   _, H = signal.freqz(h, worN=w)
   plt.figure()
   plt.plot(w, 20 * np.log10(np.abs(H)))
   plt.xlabel("Angular Frequency (rad/sample)")
   plt.ylabel("Amplitude")
   plt.savefig(f"plots/{prefix}filter_tf_{idx}")

def relu(x):
   return np.maximum(0, x)

def extrap_mag(mag, mag_freqs, freqs, first_freq_idx,
   last_freq_idx):
   new_mag = np.zeros(freqs.shape)
   new_mag[first_freq_idx:last_freq_idx] = mag
   lower_zero_freq = mag_freqs[0] * 1.1
   upper_zero_freq = mag_freqs[-1] * 1.1
   lower_coeffs = np.polyfit([lower_zero_freq, mag_freqs
      [0]], [0, mag[0]], 1)
   upper_coeffs = np.polyfit([upper_zero_freq, mag_freqs
      [-1]], [0, mag[-1]], 1)
   lower_poly = np.poly1d(lower_coeffs)
   upper_poly = np.poly1d(upper_coeffs)
   for idx in range(freqs.size):
      if freqs[idx] < mag_freqs[0] and freqs[idx] >
         lower_zero_freq:
         # Lower end extrapolation
         new_mag[idx] = lower_poly(freqs[idx])
```

```
            continue
        elif freqs[idx] > mag_freqs[-1] and freqs[idx] <
            upper_zero_freq:
            # Upper end extrapolation
            new_mag[idx] = upper_poly(freqs[idx])
            continue
    return new_mag


Ntaps = 128

parser = argparse.ArgumentParser(description='Create␣a␣
    predistorted␣waveform.')
parser.add_argument('--txwave', help="File with waveform
    that was ideally transmitted")
parser.add_argument('--rxwave', help="File with waveform
    that was recieved")
parser.add_argument('--goalwave', help="The waveform that we
     want to transmit")
parser.add_argument('--outprefix', help="Prefix for output
    files")
args = parser.parse_args()

tx_wave_dfile = Path("data", args.txwave)
rx_wave_dfile = Path("data", args.rxwave)
goal_wave_dfile = Path("data", args.goalwave)

outprefix = ""
if args.outprefix is not None:
    outprefix = str(args.outprefix)

wave_data = np.load(tx_wave_dfile)
chirp             = wave_data["chirp"]
waveform          = wave_data["waveform"]
sample_times      = wave_data["sample_times"]
chirp_bw          = wave_data["chirp_bw"]
chirp_sample_count = wave_data["chirp_sample_count"]
pri_sample_count  = wave_data["pri_sample_count"]
required_transfer_units = wave_data["required_transfer_units
    "]
sample_rate       = wave_data["sample_rate"]
instantaneous_freq = wave_data["instantaneous_freq"]
wave_data.close()

goal_data = np.load(goal_wave_dfile)
```

```
ideal_chirp = goal_data["chirp"]
goal_data.close()

print("Calculating TX chirp DFT")
chirp_dft = fftshift(fft(chirp))
chirp_freq = fftshift(fftfreq(chirp_dft.size, 1/sample_rate)
   )
chirp_freq_norm = fftshift(fftfreq(chirp_dft.size))
chirp_freq_angular_norm = chirp_freq_norm * 2 * np.pi

dft_matrix = np.vander(np.exp(-1j * chirp_freq_angular_norm)
   , N=Ntaps)

chirp_first_freq_idx = np.argmax(chirp_freq >= -chirp_bw / 2
    * 0.95 )
chirp_last_freq_idx = np.argmax(chirp_freq >= chirp_bw / 2 *
    0.95)

chirp_dft_in_bw = chirp_dft[chirp_first_freq_idx:
   chirp_last_freq_idx]
chirp_freq_in_bw = chirp_freq[chirp_first_freq_idx:
   chirp_last_freq_idx]

num_pulses_to_use = 1

for idx in range(num_pulses_to_use):
   rx_chirp = np.load(f"data/rx-chirps/{outprefix}chirp_{idx
      }.npy")

   plt.figure()
   plt.plot(range(chirp_sample_count), rx_chirp.real, label
      ="Real")
   plt.plot(range(chirp_sample_count), rx_chirp.imag, label
      ="Imaginary")
   plt.xlabel("Sample index")
   plt.ylabel("Amplitude")
   plt.legend(loc="upper right")
   plt.savefig(f"plots/{outprefix}cvx_rx_chirp_{idx}.png")

   plt.figure()
   plt.plot(range(chirp_sample_count), np.abs(rx_chirp))
   plt.xlabel("Sample index")
   plt.ylabel("Amplitude")
   plt.savefig(f"plots/{outprefix}cvx_rx_chirp_mag_{idx}.png
      ")
```

```
rx_chirp_dft = fftshift(fft(rx_chirp))
rx_chirp_dft_in_bw = rx_chirp_dft[chirp_first_freq_idx:
   chirp_last_freq_idx]
tf_est = rx_chirp_dft_in_bw / chirp_dft_in_bw
tf_est = tf_est / np.mean(np.abs(tf_est))
tf_est_db = 20 * np.log10(np.abs(tf_est))
knots = np.array([-0.7, -0.3, -0.12, 0.12, 0.3, 0.7]) *
   chirp_freq_in_bw[-1]
spline = interpolate.LSQUnivariateSpline(chirp_freq_in_bw
   , tf_est_db, knots)
spline_pts = spline(chirp_freq_in_bw)

extended_spline = extrap_mag(spline_pts, chirp_freq_in_bw
   , chirp_freq, chirp_first_freq_idx,
   chirp_last_freq_idx)

tf_mag = np.power(10, extended_spline / 20)
angular_freq = 2 * np.pi * fftshift(fftfreq(chirp_dft.
   size))
system_delay = 0
tf_est = 100 * tf_mag * np.exp(-1j * angular_freq *
   system_delay)

K = Ntaps / 2
tf_des = 100 * np.exp(-1j * angular_freq * (system_delay
   + K))

x = cp.Variable(Ntaps, complex=True)
Hfir = dft_matrix @ x
Hcas = cp.multiply(tf_est, Hfir)
Herr = cp.abs(Hcas - tf_des)
objective = cp.Minimize(cp.max(Herr))
problem = cp.Problem(objective)
problem.solve()

h = x.value

dbode(h, outprefix, idx)

# predistorted_chirp = (ideal_chirp *
   chirp_correction_window).astype(np.complex64)
predistorted_chirp = signal.convolve(ideal_chirp, h, "
   same").astype(np.complex64)
predistorted_waveform = np.array([0]*waveform.size, np.
   complex64)
predistorted_waveform[0:chirp_sample_count] =
   predistorted_chirp
plt.figure()
fig, (ax1, ax2) = plt.subplots(2, 1)
ax1.stem(np.abs(h))
ax1.set_xlabel("Tap Index n")
ax1.set_ylabel("|h[n]|")
ax2.stem(np.angle(h))
ax2.set_xlabel("Tap Index n")
ax2.set_ylabel("arg\{h[n]\}")
plt.savefig(f"plots/{outprefix}cvx_taps_{idx}.png")


plt.figure()
plt.plot(predistorted_chirp.real, label="Real")
plt.plot(predistorted_chirp.imag, label="Imaginary")
plt.xlabel("Sample index")
plt.ylabel("Amplitude")
plt.legend(loc="upper right")
plt.savefig(f"plots/{outprefix}cvx_predistorted_chirp_{
   idx}.png")
np.savez(f"data/predistorted_waveform_{idx}.npz",
       chirp=predistorted_chirp,
       waveform=predistorted_waveform,
       sample_times=sample_times,
       chirp_bw=chirp_bw,
       chirp_sample_count=chirp_sample_count,
       pri_sample_count=pri_sample_count,
       required_transfer_units=required_transfer_units,
       sample_rate=sample_rate,
       instantaneous_freq=instantaneous_freq)
```

The segment below shows the code used to implement the windowing predistortion algorithm:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal
from scipy import interpolate
from scipy.fftpack import fft, fftshift, fftfreq
import argparse
from pathlib import Path
import seaborn as sns

sns.set_theme(context="paper", style="whitegrid")
```

```
parser = argparse.ArgumentParser(description='Create␣a␣
   predistorted␣waveform.')
parser.add_argument('--txwave', help="File with waveform
   that was ideally transmitted")
parser.add_argument('--rxwave', help="File with waveform
   that was recieved")
parser.add_argument('--goalwave', help="The waveform that we
    want to transmit")
parser.add_argument('--outprefix', help="Prefix for output
   files")
args = parser.parse_args()

tx_wave_dfile = Path("data", args.txwave)
rx_wave_dfile = Path("data", args.rxwave)
goal_wave_dfile = Path("data", args.goalwave)

outprefix = ""
if args.outprefix is not None:
   outprefix = str(args.outprefix)

wave_data = np.load(tx_wave_dfile)
chirp             = wave_data["chirp"]
waveform          = wave_data["waveform"]
sample_times      = wave_data["sample_times"]
chirp_bw          = wave_data["chirp_bw"]
chirp_sample_count = wave_data["chirp_sample_count"]
pri_sample_count  = wave_data["pri_sample_count"]
required_transfer_units = wave_data["required_transfer_units
   "]
sample_rate       = wave_data["sample_rate"]
instantaneous_freq = wave_data["instantaneous_freq"]
wave_data.close()

goal_data = np.load(goal_wave_dfile)
ideal_chirp = goal_data["chirp"]
goal_data.close()

print("Calculating TX chirp DFT")
chirp_dft = fftshift(fft(chirp))
chirp_freq = fftshift(fftfreq(chirp_dft.size, 1/sample_rate)
   )

chirp_first_freq_idx = np.argmax(chirp_freq >= -chirp_bw / 2
    * 0.95 )
```

```
chirp_last_freq_idx = np.argmax(chirp_freq >= chirp_bw / 2 *
    0.95)

chirp_dft_in_bw = chirp_dft[chirp_first_freq_idx:
   chirp_last_freq_idx]
chirp_freq_in_bw = chirp_freq[chirp_first_freq_idx:
   chirp_last_freq_idx]

num_pulses_to_use = 3

for idx in range(num_pulses_to_use):
   rx_chirp = np.load(f"data/rx-chirps/{outprefix}chirp_{idx
      }.npy")

   plt.figure()
   plt.plot(range(chirp_sample_count), rx_chirp.real, label
      ="Real")
   plt.plot(range(chirp_sample_count), rx_chirp.imag, label
      ="Imaginary")
   plt.legend(loc="upper right")
   plt.xlabel("Sample index")
   plt.ylabel("Amplitude")
   plt.savefig(f"plots/{outprefix}rx_chirp_{idx}.jpg")

   plt.figure()
   plt.plot(range(chirp_sample_count), np.abs(rx_chirp))
   plt.xlabel("Sample index")
   plt.ylabel("Amplitude")
   plt.savefig(f"plots/{outprefix}rx_chirp_mag_{idx}.jpg")

   rx_chirp_dft = fftshift(fft(rx_chirp))
   rx_chirp_dft_in_bw = rx_chirp_dft[chirp_first_freq_idx:
      chirp_last_freq_idx]
   tf_est = rx_chirp_dft_in_bw / chirp_dft_in_bw
   tf_est = tf_est / np.mean(np.abs(tf_est))
   tf_est_db = 20 * np.log10(np.abs(tf_est))
   knots = np.array([-0.7, -0.3, -0.12, 0.12, 0.3, 0.7]) *
      chirp_freq_in_bw[-1]
   spline = interpolate.LSQUnivariateSpline(chirp_freq_in_bw
      , tf_est_db, knots)
   spline_pts = spline(chirp_freq_in_bw)
   plt.figure()
   plt.plot(chirp_freq_in_bw / 1e6, 20*np.log10(np.abs(
      tf_est)))
   plt.plot(chirp_freq_in_bw / 1e6, spline_pts)
```

```
    plt.plot(chirp_freq_in_bw / 1e6, -spline_pts)
    plt.xlabel("Frequency (MHz)")
    plt.ylabel("Amplitude")
    plt.savefig(f"plots/{outprefix}tf_and_splines_{idx}.jpg")
    chirp_correction_window = np.power(10, -spline(
        instantaneous_freq) / 20)
    chirp_correction_window_in_bw = np.power(10, -spline(
        chirp_freq_in_bw) / 20)
    np.clip(chirp_correction_window,
        chirp_correction_window_in_bw.min(),
        chirp_correction_window_in_bw.max(), out=
        chirp_correction_window)
    plt.figure()
    plt.plot(chirp_correction_window)
    plt.xlabel("Sample index")
    plt.ylabel("Amplitude")
    plt.savefig(f"plots/{outprefix}chirp_correction_window_{
        idx}.jpg")
    chirp_correction_window = chirp_correction_window /
        chirp_correction_window.max()
    predistorted_chirp = (ideal_chirp *
        chirp_correction_window).astype(np.complex64)
    predistorted_waveform = np.array([0]*waveform.size, np.
        complex64)
    predistorted_waveform[0:chirp_sample_count] =
        predistorted_chirp
    np.savez(f"data/predistorted_waveform_{idx}.npz",
            chirp=predistorted_chirp,
            waveform=predistorted_waveform,
            sample_times=sample_times,
            chirp_bw=chirp_bw,
            chirp_sample_count=chirp_sample_count,
            pri_sample_count=pri_sample_count,
            required_transfer_units=required_transfer_units,
            sample_rate=sample_rate,
            instantaneous_freq=instantaneous_freq)
```

The segment below shows the code used to calculate the figures of merit of interest in this thesis:

```
def nextpow2(x):
    return ceil(log2(abs(x)))

def lfm_chirp(bandwidth, duration, center_frequency=0,
    sample_spacing=1):
    chirp_sample_count = ceil(duration / sample_spacing)
```

```
   time = np.arange(0, chirp_sample_count) * sample_spacing
   chirp_rate = bandwidth / duration
   start_frequency = center_frequency - bandwidth / 2
   chirp_phase = 2 * np.pi * (chirp_rate / 2 * time**2 +
      start_frequency * time)
   return np.exp(1j * chirp_phase)

def effective_bandwidth(waveform, sample_spacing):
   waveform_dft = fftshift(fft(waveform))
   waveform_freq = fftshift(fftfreq(waveform_dft.size,
      sample_spacing))
   waveform_dft_power = np.abs(waveform_dft)**2
   moment_integrand = (2 * np.pi * waveform_freq)**2 *
      waveform_dft_power
   second_moment = np.trapz(moment_integrand, waveform_freq)
   waveform_energy = np.trapz(waveform_dft_power,
      waveform_freq)
   return np.sqrt(second_moment / waveform_energy)

def estimate_sidelobe_level(waveform, ideal_waveform=None,
   sample_spacing=1, Nresampled=None, make_plots=False):
   if ideal_waveform is None:
      waveform_corr = np.correlate(waveform, waveform, "full
         ")
   else:
      waveform_corr = np.correlate(waveform, ideal_waveform,
          "full")

   time = np.arange(waveform_corr.size) * sample_spacing

   if Nresampled is None:
      Nresampled = 2**nextpow2(waveform_corr.size * 10)

   resampled_corr, resampled_time = signal.resample(
      waveform_corr, Nresampled, t=time)
   resampled_mag = np.abs(resampled_corr)
   resampled_mag /= resampled_mag.max()
   resampled_db = 20 * np.log10(resampled_mag)

   # Identify the peaks in the pulsed compressed waveform,
      using the magnitude
   # in dB.
   peaks, _ = signal.find_peaks(resampled_db, height=-40,
      prominence=5)
   peak_heights = resampled_db[peaks]
```

```
# Sort the peaks by height, and identify the main lobe's
   location.
sorted_peaks = np.flip(np.argsort(peak_heights))
main_lobe_idx = peaks[sorted_peaks[0]]

# Identify the 1st sidelobe in either direction.
sorted_sidelobes = sorted_peaks[1:]
for idx in sorted_sidelobes:
   if peaks[idx] < main_lobe_idx:
      first_left_sidelobe_idx = peaks[idx]
      break
for idx in sorted_sidelobes:
   if peaks[idx] > main_lobe_idx:
      first_right_sidelobe_idx = peaks[idx]
      break

# Identify the location of the null on each side of the
   main lobe.
main_lobe_left_bound = np.argmin(resampled_db[
   first_left_sidelobe_idx:main_lobe_idx]) +
   first_left_sidelobe_idx
main_lobe_right_bound = np.argmin(resampled_db[
   main_lobe_idx:first_right_sidelobe_idx]) +
   main_lobe_idx

# Calculate the power in the sidelobes and main lobe.
resampled_power = resampled_mag**2
left_sidelobe_power = np.trapz(resampled_power[0:
   main_lobe_left_bound], resampled_time[0:
   main_lobe_left_bound])
right_sidelobe_power = np.trapz(resampled_power[
   main_lobe_right_bound:], resampled_time[
   main_lobe_right_bound:])
main_lobe_power = np.trapz(resampled_power[
   main_lobe_left_bound:main_lobe_right_bound],
   resampled_time[main_lobe_left_bound:
   main_lobe_right_bound])
sidelobe_power = left_sidelobe_power +
   right_sidelobe_power
islr = 10 * np.log10(sidelobe_power / main_lobe_power)

# Identify the peak sidelobe level
peak_sll = np.sort(peak_heights)[-2]
```

```
    if make_plots:
        plt.plot(resampled_time, resampled_db)
        plt.scatter(resampled_time[main_lobe_idx],
            resampled_db[main_lobe_idx])
        plt.scatter(resampled_time[first_left_sidelobe_idx],
            resampled_db[first_left_sidelobe_idx])
        plt.scatter(resampled_time[first_right_sidelobe_idx],
            resampled_db[first_right_sidelobe_idx])
        plt.scatter(resampled_time[main_lobe_left_bound],
            resampled_db[main_lobe_left_bound])
        plt.scatter(resampled_time[main_lobe_right_bound],
            resampled_db[main_lobe_right_bound])
        plt.show()

    return peak_sll, islr

def estimate_main_lobe_width(waveform, ideal_waveform=None,
    sample_spacing=1, cutoff_level=-3, Nresampled=None,
    make_plots=False):
    if ideal_waveform is None:
        waveform_corr = np.correlate(waveform, waveform, "full
            ")
    else:
        waveform_corr = np.correlate(waveform, ideal_waveform,
             "full")

    time = np.arange(waveform_corr.size) * sample_spacing

    if Nresampled is None:
        Nresampled = 2**nextpow2(waveform_corr.size * 10)

    resampled_corr, resampled_time = signal.resample(
        waveform_corr, Nresampled, t=time)
    resampled_mag = np.abs(resampled_corr)
    resampled_db = 20 * np.log10(resampled_mag)
    resampled_db -= resampled_db.max()

    zero_crossings = np.where(np.diff(np.signbit(resampled_db
         - cutoff_level)))[0]
    idx = zero_crossings[0]
    func = interpolate.interp1d([resampled_time[idx],
        resampled_time[idx+1]], [resampled_db[idx] -
        cutoff_level, resampled_db[idx+1] - cutoff_level],
        fill_value="extrapolate")
    lobe_start = optimize.fsolve(func, resampled_time[idx])
```

```
   idx = zero_crossings[1]
   func = interpolate.interp1d([resampled_time[idx],
      resampled_time[idx+1]], [resampled_db[idx] -
      cutoff_level, resampled_db[idx+1] - cutoff_level],
      fill_value="extrapolate")
   lobe_stop = optimize.fsolve(func, resampled_time[idx])

   beamwidth = lobe_stop - lobe_start

   if make_plots:
      plt.figure()
      plt.plot(resampled_time, resampled_db)
      plt.vlines([lobe_start, lobe_stop], -40, 0)
      plt.scatter(resampled_time[zero_crossings],
         resampled_db[zero_crossings])
      plt.show()

   return beamwidth

speed_of_light = 2.998e8

parser = argparse.ArgumentParser(description='')
parser.add_argument('--chirpcount', help="number of chirps
   to process")
args = parser.parse_args()

beamwidths = []
sidelobe_levels = []

effective_bandwidths = []
peak_slls = []
islrs = []

tx_wave_dfile = Path(data_dir, "waveform.npz")

wave_data = np.load(tx_wave_dfile)
chirp             = wave_data["chirp"]
waveform          = wave_data["waveform"]
sample_times      = wave_data["sample_times"]
chirp_bw          = wave_data["chirp_bw"]
chirp_sample_count = wave_data["chirp_sample_count"]
pri_sample_count  = wave_data["pri_sample_count"]
required_transfer_units = wave_data["required_transfer_units
   "]
sample_rate       = wave_data["sample_rate"]
```

```
instantaneous_freq = wave_data["instantaneous_freq"]
wave_data.close()

sample_spacing = 1/sample_rate

for chirp_num in range(int(args.chirpcount)):
   goal_wave_dfile = Path(data_dir, "waveform.npz")
   rx_wave_dfile = Path(data_dir, "rx-chirps", f"iter{
      chirp_num}_chirp_0.npy")

   goal_data = np.load(goal_wave_dfile)
   ideal_chirp = goal_data["chirp"]
   goal_data.close()

   rx_chirp = np.load(rx_wave_dfile)
   pc_chirp = signal.correlate(rx_chirp, ideal_chirp)

   lobe_width = estimate_main_lobe_width(rx_chirp,
      ideal_chirp, sample_spacing)
   eff_bw = effective_bandwidth(rx_chirp, sample_spacing)
   peak_sll, islr = estimate_sidelobe_level(rx_chirp,
      ideal_chirp, sample_spacing)

   beamwidths.append(speed_of_light * lobe_width / 2)
   effective_bandwidths.append(eff_bw / (2 * np.pi) / 1e6)
   peak_slls.append(peak_sll)
   islrs.append(islr)
```

Appendix B

Summary of Acronyms

A summary of acronyms used in this work is provided in Table B.1.

Table B.1: Acronyms used throughout this paper.

Acronym	Meaning
ADC	Analog-to-digital converter
AXI	Advanced eXtensible Interface
CPU	Central processing unit
C-SWaP	Cost, size, weight, and power consumption
DAC	Digital-to-analog converter
DDR (RAM)	Double Data Rate Synchronous Dynamic Random-Access Memory
DMA	Direct memory access
DSP	Digital signal processing
EcoSAR	Ecological Synthetic Aperture Radar
FFT	Fast Fourier transform
FIR	Finite impulse response
FPGA	Field-programmable gate array
HPA	High-power amplifier
HTTP	Hypertext Transfer Protocol
JEDEC	A semiconductor industry group that provides many widely adopted standards
JESD204B	A high-speed data converter interface standardized by the JEDEC committee
JSON	JavaScript Object Notation
LVDS	Low-voltage differential signaling
NASA	National Aeronautics and Space Administration
PC	Personal computer
SDR	Software-defined radio
SESAR	Space Exploration Synthetic Aperture Radar
UART	Universal asynchronous receiver-transmitter

Acronym	Meaning
UDP	User Datagram Protocol