A GNU Radio Based Software-Defined Radar

Lee K. Patton

Wright State University

Follow this and additional works at: https://corescholar.libraries.wright.edu/etd_all

Part of the Electrical and Computer Engineering Commons

Repository Citation

This Thesis is brought to you for free and open access by the Theses and Dissertations at CORE Scholar. It has been accepted for inclusion in Browse all Theses and Dissertations by an authorized administrator of CORE Scholar. For more information, please contact library-corescholar@wright.edu.
A GNU Radio Based Software-Defined Radar

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Engineering

by

Lee K. Patton
Department of Electrical Engineering
Wright State University

2007
Wright State University

__________________________________________
Brian D. Rigling, Ph.D.
Thesis Director

__________________________________________
Fred Garber, Ph.D.
Department Chair

Committee on
Final Examination

__________________________________________
Brian D. Rigling, Ph.D.

__________________________________________
Fred Garber, Ph.D.

__________________________________________
Edmund Zelnio, M.S.

__________________________________________
Joseph F. Thomas, Jr., Ph.D.
Dean, School of Graduate Studies
GNU Radio is an open source software-defined radio project, and the Universal Software Radio Peripheral (USRP) is hardware designed specifically for use with GNU Radio. Together, these two technologies have been used to implement very sophisticated, yet low cost, software-defined radios. Since software-defined radio and software-defined radar are really one in the same technologies, it stands to reason that GNU Radio and the USRP could be utilized to form a low cost radar sensor. In this thesis, we discuss the design of a prototype software-defined radar, built using the open source GNU Radio and open specification USRP projects. The prototype design is introduced, followed by the results of laboratory testing. A discussion on the expected operational performance of the prototype is then provided. The thesis concludes with the development and analysis of a waveform optimization algorithm that is capable of improving signal to interference plus noise ratio in the presence of a band-limited interferer. The low computational complexity of this algorithm make it amenable to software-defined radar.
Contents

I A Low Cost Software-Defined Radar 1

1 Introduction 3

1.1 Low Cost Software-Defined Radar Sensors 3
1.2 Software-Defined Radio 4
1.3 Currently Realizable SDRs 6
1.4 An Open Source SDR 7

2 GNU Radio and the USRP 9

2.1 The Universal Software Radio Peripheral 9

2.1.1 The Big Picture 9
2.1.2 The USRP 13
2.1.3 Daughterboards 18
2.1.4 The RFX2400 Transceiver 19
2.1.5 Alternate Hardware Configurations 25

2.2 GNU Radio 25

3 A GNU Radio and USRP Based Software-Defined Radar 29

3.1 Requirements 29

3.1.1 Hardware Requirements 29
5.3.1 Dead Zone .................................................. 80
5.4 Looking Toward the Future .................................... 82
5.5 Conclusion ...................................................... 82

Bibliography ....................................................... 83

II Nonquadratic Regularization for Waveform Optimization 85

6 Nonquadratic Regularization for Waveform Optimization 87
  6.1 Introduction .................................................. 88
  6.2 Development .................................................. 89
    6.2.1 Signal Model ........................................... 89
    6.2.2 Matched Filter SINR .................................... 90
    6.2.3 Cost Function ........................................... 92
    6.2.4 Gradient Descent Algorithm .............................. 93
    6.2.5 Stochastic Gradient Descent Algorithm ................. 94
    6.2.6 Application ............................................. 94
  6.3 Simulation .................................................. 96
    6.3.1 Implementation .......................................... 96
    6.3.2 Adaptation of a Short Pulse ............................ 96
    6.3.3 Adaptation of Other Waveforms ......................... 97
  6.4 Conclusion .................................................. 100

Bibliography ....................................................... 103
List of Figures

1.1 Block diagram of an ideal software-defined radio communication system . . 4
1.2 Block diagram of a currently realizable software-defined radio communi-
cation system . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

2.1 Block diagram of an SDR currently realizable using GNU Radio, the USRP,
and associated daughterboards . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10
2.2 USRP motherboard without daughterboard . . . . . . . . . . . . . . . . . . . . 11
2.3 The USRP motherboard with basic transmit and receive daughterboards . . 12
2.4 USRP receive and transmit signal paths . . . . . . . . . . . . . . . . . . . 14
2.5 The digital down-conversion (DDC) and decimation stage in the USRP
receive path. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
2.6 The digital up-conversion (DUC) in the USRP transmit path. . . . . . . . . 15
2.7 Receive signal path for the RFX2400 daughterboard . . . . . . . . . . . . 21
2.8 Transmit signal path for the RFX2400 daughterboard . . . . . . . . . . . . 22
2.9 The RFX2400 transceiver . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23

3.1 Block diagram of radar software functionality . . . . . . . . . . . . . . . . 32
3.2 Block diagram of software-defined radar in time synchronization calibra-
tion mode . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 34
3.3 Block diagram of software-defined radar in operational mode . . . . . . . 34
4.15 Actual matched filter response of a length-5 Barker sequence (1 samples/chip) with no pre-filtering to remove system effects

4.16 Actual matched filter response of a length-5 Barker sequence (2 samples/chip) with no pre-filtering to remove system effects

4.17 Normalized frequency response

4.18 Impulse response

4.19 Normalized frequency response inverse filter

4.20 Simulated effect of pre-filtering transmit waveform with inverse filter

4.21 Actual matched filter response of a length-5 Barker sequence (1 samples/chip) when pre-filtering has been applied to remove system effects

4.22 Actual matched filter response of a length-5 Barker sequence (2 samples/chip) when pre-filtering has been applied to remove system effects

4.23 Phase coherence plot of received signal with no pre-filtering to remove system effects

4.24 Close-up of Figure 4.23

4.25 Phase coherence plot of received signal with pre-filtering to remove system effects

4.26 Close-up of Figure 4.25

4.27 Hardware configuration for radar simulator

5.1 Simplified block diagram of USRP transmit/receive system that illustrates key components in power budget calculations

5.2 SNR vs. target range and radar cross section

5.3 Dead zone (km) vs. signal processing gain and bandwidth

6.1 Power spectral density of adapted short pulse for each implementation with no regularization
6.2 SINR during adaptation of a short pulse for each implementation with no regularization. See Figure 6.3 for corresponding matched filter responses. 97

6.3 Matched filter response after adaptation of a short pulse for each implementation with no regularization. See Figure 6.2 for corresponding SINR. 98

6.4 SINR of adapted short pulse for GD implementation with various $p = 1$ regularization weights. See Figure 6.5 for corresponding matched filter responses. 98

6.5 Matched filter response of adapted short pulse for GD implementation with various $p = 1$ regularization weights. See Figure 6.4 for corresponding SINR. 98

6.6 SINR of adapted short pulse for ISG implementation with various $p = 1$ regularization weights. See Figure 6.7 for corresponding matched filter responses. 99

6.7 Matched filter response of adapted short pulse for ISG implementation with various $p = 1$ regularization weights. See Figure 6.6 for corresponding SINR. 99

6.8 Matched filter response of adapted LFM waveform for each implementation with no regularization. See Figure 6.2 for corresponding SINR. 100

6.9 SINR of adapted LFM waveform for GD implementation with various $p = 1$ regularization weights. See Figure 6.10 for corresponding matched filter responses. 101

6.10 Matched filter response of adapted LFM waveform for GD implementation with various $p = 1$ regularization weights. See Figure 6.9 for corresponding SINR. 101

6.11 SINR of adapted LFM waveform for ISG implementation with various $p = 1$ regularization weights. See Figure 6.12 for corresponding matched filter responses. 101
6.12 Matched filter response of adapted LFM waveform for ISG implementation with various $p = 1$ regularization weights. See Figure 6.11 for corresponding SINR. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 102
List of Tables

2.1 USRP daughterboards currently available from Ettus Research . . . . . 19
2.2 Gain and noise figures of select RFX2400 components . . . . . . . . . 24
4.1 Waveforms used to test radar transmission . . . . . . . . . . . . . . . 56
5.1 Radar range equation parameters . . . . . . . . . . . . . . . . . . . . 79
5.2 Dead zone (km) vs. signal processing gain and bandwidth. . . . . . . 81
Acknowledgment

This project has been made possible in part due to funding from two branches of the Air Force Research Laboratory’s Sensors Directorate, Automatic Target Recognition Division. These branches are: the Automatic Target Recognition Branch (AFRL/SNAT) and the Find, Fix, Track & ID Branch (AFRL/SNAR). I would like to thank these organizations for their support. I would also like to thank those individuals who have contributed to this work. Namely, I would like to thank David Kuhl of ATK-Mission Research Corporation for the use of his time, advice and personal RF delay lines; Mark Stewart for doing all of the soldering work required to modify the USRP and RFX2400 for coherent operation; Eric Blossom and Matt Ettus for their personal correspondence about this project; and all those who have responded to my questions on the GNU Radio mailing list. Lastly, I would like to thank my wife Kristen for her loving patience and support.
To Kristen and Ellie.
Part I

A Low Cost Software-Defined Radar
Chapter 1

Introduction

1.1 Low Cost Software-Defined Radar Sensors

GNU Radio is an open source software-defined radio project, and the Universal Software Radio Peripheral (USRP, pronounced “usurp”) is hardware designed specifically for use with the GNU Radio software. Together, these two technologies have been used to implement very sophisticated, yet low cost, software-defined radios. Since software-defined radio and software-defined radar are really one in the same technologies, it stands to reason that GNU Radio and the USRP could be utilized to form a low cost radar sensor. Hence, there are two questions this thesis attempts to answer: (1) Can the GNU Radio and USRP be utilized as a software-defined radar sensor? (2) If so, what performance can be expected from the system? The results of this investigation are the subject of Part I of this thesis. Part II of this thesis discusses a low-complexity waveform optimization algorithm amenable to software-defined radar applications.

This chapter begins with an introduction to software-defined radio. The GNU Radio and USRP projects are then introduced. Next, the USRP transmit and receive signal paths are discussed. This is followed by a discussion of the radio frequency front-ends used in conjunction with the USRP. The chapter concludes with an explanation of the GNU Radio
1.2 Software-Defined Radio

A software-defined radio (SDR) is a radio communication system that performs radio signal modulation and demodulation in software. [1] The exact extent to which a particular radio system may be considered an SDR is not entirely clear because the SDR community has not yet formulated definitive answers to questions regarding the types of processor upon which the software can run, or the percentage of total signal processing that must be performed in software. However, it is clear that the philosophy behind the SDR concept is that the software should operate as close to the antenna as possible (in terms of intervening signal processing stages), and this software should run on a general purpose computer. [2] See Figure 1.1 for the block diagram of an ideal SDR.

![Figure 1.1: Block diagram of an ideal software-defined radio communication system. In this system, the antennas, ADC/DACs, and computer are capable of processing any radio signal of interest in real-time.](image)

The impetus of SDR research is a desire to shift the radio engineering problem from the hardware domain to the software domain. The advantage of this problem-space translation
is that the software domain provides an inherently more flexible, predictable, repeatable, and accessible solution space than does the hardware domain. In the software domain, all radios are differentiated solely by the software required to implement them. Therefore, a single SDR system could become one of any number of RF transceivers (e.g., GPS, 802.11, HDTV) by simply executing a different block of code residing in memory.

Of course, realizing the ideal SDR of Figure 1.1 means overcoming some formidable obstacles. Consider the components of this system:

1. Ideal transmit and receive antennas – These antennas can operate at the carrier frequencies of all radio signals of interest.

2. Ideal analog-to-digital and digital-to-analog converters – These devices have sampling rates greater than two times the carrier frequency of all radio signals of interest.\(^1\) At this sampling rate, all signals of interest can be processed at their carrier frequency.

3. Ideal computer – This computer has sufficient processing power to handle the real-time signal processing and protocol management demands for all radio signals of interest.

In reality, antennas must be designed for operation within a particular frequency band, modern analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) are not fast enough to process a large portion of the occupied spectrum, and current general purpose computers are still not sufficient to handle the real-time demands of many applications.

---

\(^1\)The Nyquist sampling theorem states a signal must be sampled at a rate greater than two times its highest frequency component in order to avoid aliasing. [3, pp.321]
1.3 Currently Realizable SDRs

The SDR of Figure 1.1 has been realized for some relatively simple and low frequency applications such as AM and FM radio. However, many applications of interest cannot yet be realized using this configuration. Instead, a different configuration is being used that employ field-programmable gate arrays (FPGAs) and superheterodyne mixing stages, called the RF front ends, to augment system performance. This is shown in Figure 1.2

![Block diagram of a currently realizable software-defined radio communication system](image)

In the receiver, the RF front end translates the received signal from its carrier frequency to an intermediate frequency (IF) or to baseband. In the transmitter, the RF front end translates the transmit signal from IF or baseband to the desired carrier frequency. In either case, the ADC/DAC need only convert the signal over its modulation bandwidth, and not the entire bandwidth from DC to carrier.\(^2\)

The use of lower data rate ADC/DACs helps to lessen the computational burden of the computer. However, processing requirements are often still too demanding. To diminish the computational burden of the computer even further, an FPGA can be positioned between the

\(^2\)This is true at baseband. At IF, the system must convert the signal from DC to IF plus the upper half of the modulation bandwidth, but even in this case a lower rate ADC/DAC can be used.
ADC/DAC and the computer. The FPGA performs the computationally expensive signal processing that is common to all SDRs. Examples of such processing includes digital down/up-conversion and decimation/interpolation filtering.

While the augmented system of Figure 1.2 is realizable today, it is not as flexible as the ideal SDR. The introduction of the FPGA does not significantly affect system flexibility because its functionality is common to all radio types. However, all RF front ends (including the antenna) are limited to operation within a particular frequency band. Therefore, the radio types an SDR augmented with an RF front end can become is limited to those radios with signals in the operational bandwidth of the front end.

1.4 An Open Source SDR

GNU Radio (GR) is an open source software project founded by Mr. Eric Blossom, who wanted to create a software-based high-definition television (HDTV) receiver in advance of broadcast flag legislation limiting the types of hardware allowed to receive the HDTV signal. [4] Today, the GNU Radio project has grown into an architecture for implementing various SDRs. The software is comprised of hardware-independent signal processing code and hardware-dependent interface code, which provides the link between the signal processing code running on a general purpose computer and the actual radio hardware (i.e., ADC/DAC or FPGA).

The list of radio hardware with currently implemented GR interfaces includes personal computer audio cards and television tuner cards, as well as high-end ADCs. However, the hardware most widely used in conjunction with GR is the Universal Software Radio Peripheral (USRP, pronounced “usurp”). In fact, the block diagram shown in Figure 1.2 is the block diagram of the SDR formed when GNU Radio software running on a general purpose computer is used to communicate with a USRP.

---

[A more complete list may be found in [5]]
Chapter 2

GNU Radio and the USRP

2.1 The Universal Software Radio Peripheral

2.1.1 The Big Picture

The first USRP was designed and built by Mr. Matt Ettus, who secured National Science Foundation funding for the project through the University of Utah. [4] Currently, the USRP is in its fourth revision, and can be purchased from Ettus Research, LLC. The best description of the USRP probably comes from The USRP User’s and Developer’s Guide, which says:

The [USRP] is designed to allow general purpose computers to function as high bandwidth software radios. In essence, [the USRP] serves as a digital baseband and IF section of a radio communication system. In addition, it has a well-defined electrical and mechanical interface to RF front-ends (daughterboards) which can translate between that IF or baseband and the RF bands of interest. [6]

Figure 2.1 shows how the SDR of Figure 1.2 can be constructed using GNU Radio, the USRP, and associated daughterboards. The USRP provides the ADC/DAC and FPGA
functionality, while various daughterboards, also available from Ettus Research, literally snap into the USRP to provide the frequency translation functionality of the RF front end. Pictures of a USRP and basic daughterboards are shown in Figures 2.2 and 2.3. The USRP’s daughterboard interfaces are shown in Figure 2.2 labeled J66X.

![Block diagram of an SDR currently realizable using GNU Radio, the USRP, and associated daughterboards](image)

Figure 2.1: Block diagram of an SDR currently realizable using GNU Radio, the USRP, and associated daughterboards

The USRP and associated daughterboards were developed under an open specifications project. This means that the computer-aided design (CAD) files used in their design – including schematics, Gerber files, and bills-of-material – are available for download from [8]. In addition, the hardware was designed using free and open source CAD software. The schematics were created in gEDA, and the board layouts were created in PCB. [6] More information on these tools can be found at [9] and [10] respectively. The FPGA Verilog designs were compiled using the Quartus II Web Edition from Altera. [6]. This compiler is available for download at [11], and the USRP end user is free to modify the FPGA firmware.
Figure 2.2: USRP motherboard without daughterboard. From [7]
Figure 2.3: The USRP motherboard with basic transmit and receive daughterboards. From [6]
2.1.2 The USRP

The philosophy behind the USRP design is that all of the waveform-specific processing, like modulation and demodulation, should be performed on the host CPU, and all of the high-speed general purpose operations like digital up- and down-conversion, decimation, and interpolation should be performed on the FPGA. [6]. Figures 2.4-2.6 show the USRP transmit and receive signal paths. The reader is urged to compare this figure with the actual picture of the USRP in Figure 2.2, and to refer back to these figures while reading the subsequent discussion.

The Receive Signal Path

The USRP has two slots that accept receive daughterboards. These slots are labeled RxA and RxB, and correspond to interfaces J666 and J668 respectively. Each interface accepts two real-valued voltage signals from the daughterboard. These signals are labeled \( \text{VIN}_A \times \) and \( \text{VIN}_B \times \), where \( X \) is replaced by \( A \) or \( B \) depending upon which receiver slot the signals belong to. (i.e., RxA or RxB) Since both receiver slots are identical, these signals will henceforth be called \( \text{VIN}_A \) and \( \text{VIN}_B \) unless further clarification is required.

The analog signals \( \text{VIN}_A \) and \( \text{VIN}_B \) are sent to two separate ADCs. Each 12-bit ADC samples the signals at a rate of 64 mega-samples per second. The 12-bit samples are then sent to the FPGA for processing. Upon entering the FPGA, the digitized signals are routed by a multiplexer, or MUX, to the appropriate digital down-converter (DDC). A very good description of the MUX is provided in [12].

The DDC is essentially a complex mixing stage. It expects in-phase and quadrature inputs. It is up to the user to specify whether \( \text{VIN}_A \times \), \( \text{VIN}_B \times \), \( \text{VIN}_A \), \( \text{VIN}_B \), or all zeros should be routed to the in-phase or quadrature port of each of the four DDCs. Each DDC mixes its input signal to baseband. Once down-converted, the signal is decimated by a factor specified by the user. Decimation is performed in two stages. Assuming a decimation factor of \( M \) is requested by the user, the signal is first decimated by a factor of \( M/2 \) using a
Figure 2.4: USRP receive and transmit signal paths. The DDC and DUC operations are illustrated in more detail in Figures 2.5 and 2.6 respectively.
Figure 2.5: The digital down-conversion (DDC) and decimation stage in the USRP receive path.

Figure 2.6: The digital up-conversion (DUC) in the USRP transmit path.
cascaded integrator-comb (CIC) filter. The last decimation by 2 is performed by a half-band filter. The DDC and decimation stage is shown in Figure 2.5.

The in-phase and quadrature output of each half-band filter, if requested by the user, is then interleaved and pushed into a first-in-first-out (FIFO) buffer. These data samples are then sent by the USB 2.0 interface chip to the host computer for processing. Figure 2.4 shows four DDC/decimation stages. Currently, only two of these stages are implemented. This means the end user can specify two channels, and receive the data from both RxA and RxB. Each complex sample (i.e., in-phase and quadrature values) is sent using 32 bits (16 bits for in-phase, and 16-bits for quadrature). It should be noted that the USB 2.0 interface chip can transfer data at a maximum rate of 32 mega-bytes per second. This limits the bandwidth of signals that may be transferred to and from the host computer.

A Receive Signal Path Example

The operation of the USRP receive signal path might be made more clear by an example.

Consider an RFX2400 daughterboard connected to RxA, and a BasicRx daughterboard connected to RxB. (More information on these daughterboards will be presented later.) The RFX2400 is a direct-conversion daughterboard, meaning this particular RF front end translates the received signal directly to baseband from its carrier frequency. For this daughterboard, VIN_A_A represents the in-phase component of the baseband signal, and VIN_B_A represent the quadrature component. The BasicRx has two inputs, but in this example only one of them – VIN_A_B – is connected to an AM antenna.

All four signals (VIN_A_A, VIN_B_A, VIN_A_B, and VIN_B_B) are digitized by their respective ADC, and are sent to the FPGA for processing. In this example, the user has specified a MUX value such that VIN_A_A is routed to the in-phase input of DDC1,

16 bit samples (8 bits for in-phase, and 8 bits for quadrature) are also available. However, these samples are generated by simply sending the eight most significant bits of each in-phase and quadrature sample. The result is that small signals might not be detected. In addition, there is no rounding algorithm implemented for this mode. Therefore, the received signal will be noisier than when 32 bits are used.

\[
\text{For example, at 32 bits (i.e., 4 bytes) per sample, the maximum USB bandwidth in receive only mode is } 32\text{MB/sec} \div 4\text{Samples/B} = 8\text{Samples/sec} = 8\text{MHz}.
\]
VIN_B_A is routed to the quadrature input of DDC1, VIN_A_B is routed to the in-phase input of DDC0, and all zeros are routed to the quadrature input of DDC0. Note that the DDC stage is a complex mixing operation. So, even though all zeros are sent to the quadrature input of DDC0, the quadrature output of DDC0 will be nonzero, and cannot be ignored because it contains information about the in-phase input.

After digital down-conversion, the signals are decimated by an amount specified by the user. This decimated data is then interleaved and pushed into the receive FIFO. In this example, the user has requested that both channels be sent to the host computer. Therefore, the data sent over the USB to the host computer is in the format:

\[
\{I_n^{(0)}, Q_n^{(0)}, I_n^{(1)}, Q_n^{(1)}, I_{n+1}^{(0)}, Q_{n+1}^{(0)}, I_{n+1}^{(1)}, Q_{n+1}^{(1)}, \ldots\}
\]

where \(I_n^{(X)}\) and \(Q_n^{(X)}\) represent the \(n^{th}\) in-phase and quadrature samples at the output of decimator X.

**The Transmit Signal Path**

The transmit signal path works very much like the receive signal path, but in reverse. First, interleaved data sent from the host computer is pushed into the transmit FIFO on the USRP. The interleaved data has the format:

\[
\{I_n^{(0)}, Q_n^{(0)}, I_n^{(1)}, Q_n^{(1)}, I_{n+1}^{(0)}, Q_{n+1}^{(0)}, I_{n+1}^{(1)}, Q_{n+1}^{(1)}, \ldots\}
\]

where \(I_n^{(X)}\) and \(Q_n^{(X)}\) represent the \(n^{th}\) in-phase and quadrature samples intended for interpolator X. Each complex sample is 32 bits long (16-bits for in-phase, and 16-bits for quadrature). This data is de-interleaved, and sent to the input of an interpolation stage. Assuming, the user has specified an interpolation factor of L, this interpolation stage will interpolate the input data by a factor of \(L/4\) using CIC filters.

The output of the interpolation stage is sent to the demultiplexer, or DEMUX. The
DEMUX is less complicated than the receiver MUX. Here, the in-phase and quadrature output of each CIC interpolator is sent to in-phase and quadrature inputs of one of the DAC chips on the motherboard. The user specifies which DAC chip receives the output of each CIC interpolator.

Inside of the DAC, the complex-valued signal is interpolated by a factor of 4 using half-band filter interpolators. This completes the requested factor of L interpolation. After the half-band interpolators, the complex-valued signal is sent to a digital up-converter (DUC). Note, at this point the signal is not necessarily modulated to a carrier frequency. The daughterboard might further up-convert the signal.

The in-phase and quadrature output of the DUC are sent as 14 bit samples to individual digital-to-analog converters, which convert them to analog signals at a rate of 128 mega-samples per second. These analog signals are then sent from the AD9862 to either daughterboard interface J667 or J669, which represent slots TxA and TxB respectively.

2.1.3 Daughterboards

Table 2.1 lists the daughterboards currently available from Ettus Research. The Basic boards have no tuning or amplification, and are essentially interfaces to the USRP for external front ends. All other boards have tuning and amplification. Design files for each are available from [8].

Instructions for designing a daughterboard for the USRP are given in [13], which lists the electrical and mechanical interface specifications. At least one daughterboard has been fabricated by someone other than Ettus Research. In the Spring of 2006, two Wright State University undergraduates designed, built and tested a simple 5 GHz receiver daughterboard.
Table 2.1: USRP daughterboards currently available from Ettus Research

<table>
<thead>
<tr>
<th>Name</th>
<th>Functionality</th>
<th>Frequency Range (MHz)</th>
<th>Cost (USD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasicRx</td>
<td>Receiver</td>
<td>2 to 300+</td>
<td>$75.00</td>
</tr>
<tr>
<td>BasicTx</td>
<td>Transmitter</td>
<td>2 to 200</td>
<td>$75.00</td>
</tr>
<tr>
<td>LFRX</td>
<td>Receiver</td>
<td>0 to 30</td>
<td>$75.00</td>
</tr>
<tr>
<td>LFTX</td>
<td>Transmitter</td>
<td>0 to 30</td>
<td>$75.00</td>
</tr>
<tr>
<td>TVRX</td>
<td>Receiver</td>
<td>50 to 70</td>
<td>$100.00</td>
</tr>
<tr>
<td>DBSRX</td>
<td>Receiver</td>
<td>800 to 2400</td>
<td>$150.00</td>
</tr>
<tr>
<td>RFX400</td>
<td>Transceiver</td>
<td>400 to 500</td>
<td>$250.00</td>
</tr>
<tr>
<td>RFX900</td>
<td>Transceiver</td>
<td>800 to 1000</td>
<td>$275.00</td>
</tr>
<tr>
<td>RFX1200</td>
<td>Transceiver</td>
<td>1150 to 1450</td>
<td>$275.00</td>
</tr>
<tr>
<td>RFX1800</td>
<td>Transceiver</td>
<td>1500 to 2100</td>
<td>$275.00</td>
</tr>
<tr>
<td>RFX2400</td>
<td>Transceiver</td>
<td>2300 to 2900</td>
<td>$275.00</td>
</tr>
</tbody>
</table>

2.1.4 The RFX2400 Transceiver

The RFX2400 2.4 GHz transceiver daughterboard was used in radar testing, and is therefore described in detail here.

Tuning

The RFX2400 is designed around two chips from Analog Devices, the AD8347 quadrature demodulator and the AD8349 quadrature modulator. Both of these chips are direct conversion, which means they can translate signals between baseband and 2.4 GHz without the use of intermediate mixing stages. The local oscillator signals that drive the AD834* on the RFX2400 are synthesized by phase-lock loops (PLL), which is driven by a voltage-controlled oscillator (VO) operating at a fixed frequency. The Analog Devices ADF4360 is used to implement this PLL. It has a typical frequency lock time of 250 \( \mu \text{s} \), and can only take on discrete frequency values.

Tuning the RFX2400 is a two step process. First, the daughterboard is tuned as close
as possible to the desired frequency. After the daughterboard frequency has been set, the
DDC or DUC, whichever is applicable, is set to make up the difference in frequency. It
should be noted that for the RFX2400, the daughterboard is always requested to tune to a
frequency 4 MHz away from the target frequency. The explanation for this is provided as a
comment in the source code:\(^3\)

Offsetting the LO helps get the Tx carrier leakage out of the way. This also
ensures that on Rx, we’re not getting hosed by the FPGA’s DC removal loop’s
time constant. We were seeing a problem when running with discontinuous
transmission. Offsetting the LO made the problem go away.

This may be subject to change in future releases of the code.

**Transmit Power and Isolation**

Figures 2.7 and 2.8 show block diagrams of the RFX2400 receive and transmit signal paths
respectively, and Figure 2.9 shows a picture of an actual RFX2400 daughterboard. Notice
that a portion of the circuitry is common to both the transmitter and receiver, and this
circuitry is connected to the antenna port labeled \(Tx/Rx\). Ettus claims that the maximum
transmit power of the RFX2400 is greater than 13 dBm, but isolation is maintained between
the two paths connected to the Tx/Rx port by an RF switch. According to its data sheet,
the RF switch can provide approximately 22 dB of isolation at 2.4 GHz. More isolation is
expected between transmitter and the receiver port labeled \(Rx2\). Note that the receiver can
only use one port at a time.

**System Noise Figure**

The equivalent system noise figure of the daughterboard was calculated using the method
found in [14, pp. 729-732], which states the equivalent noise factor of \(N\) networks (i.e.,
\[^3\]The interested reader should see the \texttt{tune()} method in \texttt{usrp.py} and the \texttt{set_freq()} method in \texttt{db_flexrf.py}.
Both files are found in the standard GNU Radio installation.
Figure 2.7: Receive signal path for the RFX2400 daughterboard

From Tx

Antenna
Cable
G1 = L1
NF = L1

Tx/Rx

SAWTEK85591
2.4 – 2.4835 MHz
Filter
G2 = -2.75 dB
N2 = 2.75 dB

HMC174MS8
Switch
G3 = -0.6 dB
N3 = 0.6 dB

From Tx

Antenna
Cable
G1 = L1
NF = L1

Rx2

HMC174MS8
Switch
G4 = -0.6 dB
N4 = 0.6 dB

MGA82563
LNA
G5 = 12.5 dB
N5 = 2.2 dB

Rx VCO

AD8852
ADC
G9 = ?
N9 = ?

AD8347-OUT
G8 = ?
N8 = ?

20 MHz
Filter
G7 = 6 dB
N7 = 6 dB

AD8347-MIX
Quad Demod
G6 = 9 dB
N6 = 11.75 dB

ADF4360
Rx PLL

To FPGA
Figure 2.8: Transmit signal path for the RFX2400 daughterboard
Figure 2.9: The RFX2400 2.4 GHz transceiver daughterboard from Ettus Research. From [8]
components) in cascade is given by

\[ F = F_1 + \frac{F_2 - 1}{G_1} + \frac{F_3 - 1}{G_1 G_2} + \cdots + \frac{F_N - 1}{G_1 G_2 \cdots G_{N-1}} \]

where \( F_n \) and \( G_n \) are the noise factor and gain associated with component \( n \). The equivalent system noise figure is just the noise factor expressed in decibels; i.e., \( N_f = 10 \log(F) \).

Table 2.2 lists the components used in the noise figure calculation along with their associated gains and noise figures, which were determined from manufacturer data sheets. Using this data, a lower bound on RFX2400 noise figure was calculated to be 7.8 dB and 4.5 dB for the Tx/Rx and Rx2 ports respectively. These calculations do not include cable losses (labeled \( L_1 \) in Figure 2.7). A cable loss of 1 dB would increase the RFX2400 noise figures to 8.8 and 5.5 dB respectively.

<table>
<thead>
<tr>
<th>Component</th>
<th>Functionality</th>
<th>Gain (dB)</th>
<th>Noise Figure (dB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SAWTEK85591</td>
<td>RF Filter</td>
<td>-2.75</td>
<td>2.75</td>
</tr>
<tr>
<td>HMC174MS8</td>
<td>RF Switch</td>
<td>-0.6</td>
<td>0.6</td>
</tr>
<tr>
<td>MGA825632</td>
<td>LNA</td>
<td>12.25</td>
<td>2.2</td>
</tr>
<tr>
<td>AD8347-MIX</td>
<td>Mixer</td>
<td>39</td>
<td>11.75</td>
</tr>
<tr>
<td>FILTER</td>
<td>IF Filter</td>
<td>-6</td>
<td>6</td>
</tr>
</tbody>
</table>

It should be noted that no noise factor data was attainable for the components labeled \( AD8347-OUT \) and \( AD682 \). However, since additionally cascaded components can only degrade noise figure, the calculated noise figures can be considered as lower bounds on the system noise figure. And, in any reasonable receiver design, the noise figure is dominated by the components closest to the antenna. Therefore, it is expected that these lower bounds are close to the value that would be calculated if data for all components were known.
2.1.5 Alternate Hardware Configurations

Since the USRP and daughterboard CAD files are available for download, there is no limit to the modifications an end user might make to the hardware. Be that as it may, most users never modify their boards. Those that do, typically modify them in order to synchronize multiple USRPs in time, or to synchronize the local oscillators on a given daughterboard. Instructions for performing the necessary hardware modifications to achieve these goals can be found at [15]. Synchronizing the local oscillators on a given daughterboard will ensure coherence between transmit and receive. However, according to Ettus, doing so will increase the phase noise.

The output filter of the RFX2400 (labeled SAWTEK85591 in Figures 2.7 and 2.8) can also be bypassed by following the instructions found at [16]. According to Ettus, this will increase output power, improve noise figure, and increase frequency coverage.

2.2 GNU Radio

GNU Radio is an open source software framework for writing SDR applications under Linux, Unix, Windows or Mac OS X. In addition to a set of base classes from which custom applications can be derived, GNU Radio provides a host of signal processing, hardware interface, graphical user interface, and utility libraries. The GNU Radio framework is built around the graph design pattern. In each GNU Radio application, a flow graph is constructed from nodes called blocks, which are connected by edges called ports. Each block performs some aspect of the signal processing, and data transport between blocks is facilitated by connecting the input port of one block to the output port of another. Under this framework, the application programmer can create custom blocks and connect them together to form a graph. Once all the blocks have been appropriately connected, the application is sent into a loop, and the GNU Radio framework takes care of streaming data from

---

\(^4\)Currently, only binary versions of GNU Radio are available for Windows.
a data source, through the blocks, and out to a data sink.

GNU Radio is a hybrid system; the performance critical portions such as signal processing blocks are written in C++ while non-critical portions such as graph construction and policy management are written in the Python programming language. This allows developers to write highly optimized signal processing code in C++, but use the much more friendly language Python to construct applications. GNU Radio applications written in Python access the C++ signal processing blocks through interfaces automatically generated by SWIG (the Simple Wrapper Interface Generator) for Python. More details on SWIG can be found at www.swig.org. Readers interested in learning more about the GNU Radio architecture, including how to write a signal processing block, should see [17]-[19].

There are many advantages to using the GNU Radio framework to develop SDR applications, not the least of which is a very active development community and mailing list. Other key advantages of the GNU Radio framework are listed by Eric Blossom on his website [19]. His list has been reproduced here as follows:

- Hybrid C++ / Python system. The primitive signal processing blocks are implemented in C++. All graph construction, policy decisions and non-performance critical operations are performed in Python. All of the underlying runtime system is manipulable from Python.

- High Performance. Zero copy circular buffering via memory manager tricks, hand coded filtering kernels that take advantage of the SSE and 3DNow! SIMD instruction sets.

- Fixed and variable rate blocks. In addition to working with synchronous data flows, the system supports variable rate blocks. This feature is essential for variable rate compression and decompression and some synchronization strategies.

- Reconfigurable on-the-fly. The parameters of signal processing blocks may be modified at runtime. In addition, the topology of the signal processing graph itself may
be modified as needed.

- Usable as a component of standard Unix pipeline. Signal processing blocks have the ability to signal that they have finished their computation. This information is propagated through flow graph. When all blocks have finished, the runtime scheduler indicates completion.

- Graphical User Interface. GUI’s are built using any toolkit accessible from Python. We recommend the wxPython toolkit to maximize cross platform portability. Widgets are provided for visualizing data streams in the time and frequency domains (e.g., multi-channel digital oscilloscope, FFT).

- Over 100 sinks, sources and primitives. Input to and from files, TCP, high speed A/D’s, D/A’s, sound cards, all kinds of filtering, NCO’s, VCO’s, modulators, demodulators, forward error correction, etc. The libraries provided with GNU Radio are extensive, and depending upon the application, an end user might not need to write a single signal processing block.

It should be noted that GNU Radio was designed for SDR applications involving the continuous transmission or reception of radio signals. In its current form, it is very difficult to use GNU Radio to implement complicated two-way asynchronous communication systems such as TDMA and packet-based systems. However, there is significant work being done by BBN Technologies, under contract with the Defense Advanced Research and Planning Agency (DARPA), to modify the GR framework to be more amenable to asynchronous packet-based communication.\(^5\)

Chapter 3

A GNU Radio and USRP Based Software-Defined Radar

In this chapter, the design of a GNU Radio based software-defined radar prototype is discussed. The chapter begins with a discussion of the system requirements that influenced the design. This is followed by discussions of how these requirements were met in both hardware and software. The chapter concludes with a detailed analysis of the system transfer function.

3.1 Requirements

3.1.1 Hardware Requirements

The focus of this research was not the design of a new radar sensor, but an investigation of the utility of existing hardware and software for software-defined radar applications. Therefore, instead of imposing application-specific performance requirements on the sensor, the performance of the sensor hardware was evaluated in order to determine the applications for which the sensor is suitable. Modifications to the hardware were made when these

---

1Hereafter, the phrase GNU Radio may also refer to the USRP and daughterboard.
modifications were relatively easy to make, and did not violate the spirit of using these components “off-the-shelf.”

3.1.2 Software Requirements

In order to be useful as a radar sensor, the system must be capable of transmitting and receiving data such that the time between pulse transmission and reception can be known exactly. For the purposes of this thesis, we will say that such a system exhibits time-coherence and time-synchronization, which are defined as follows.

A stream of digital data samples is said to exhibit time-coherence if a time value can be assigned to each sample such that the difference in the time values assigned to any two samples is equal to the difference between the actual times at which the samples were converted either to, or from, an analog signal. That is, if the system is time-coherent, then the discrete data signal accurately represents its analog counterpart in time. Notice that this definition of time-coherence is concerned only with the duration between samples, and not the actual times at which these samples were converted. Two streams of digital data are said to lack time-synchronization if each stream is time-coherent within itself, but the two-streams are not time-coherent with respect to one another. In radar systems, time-synchronization must exist between the transmit and receive data streams.

3.2 Software Design

Achieving time-coherence and time-synchronization using GNU Radio is not exactly straightforward. While GNU Radio allows for the simultaneous transmission and reception of radio signals, it was not necessarily designed to allow the transmitter and receiver to work cooperatively, which is required in radar applications. Therefore, obtaining time-

---

2 Since the CAD files for the USRP and associated daughterboards are freely downloadable, the end-user may modify the board in any way he or she chooses. However, the spirit of this research was to use these components as close to off-the-shelf as possible. If too much effort is required for modifications, then the end-user might as well develop custom hardware.
synchronization must be addressed explicitly. Also, while the GNU Radio signal processing blocks work fast enough to handle streaming data, GNU Radio is not a real-time computing system in which the hardware and software are subject to actual time constraints. [20]. Therefore, maintaining time-coherence must also be addressed.

3.2.1 Time-Coherence

The conceptual operation of the GNU Radio radar application is illustrated in Figure 3.1. The radar transmit waveform is read in from a data file by the transmit signal processing block (Tx block), which pushes this data into a software first-in-first-out (FIFO) buffer controlled by the GR framework. The GR framework facilitates the transfer of this FIFO data over the the USB to the USRP where it is pushed into a FIFO on the FPGA. The USRP transmit signal processing path then pulls the data from this buffer as needed. An identical situation exists for the received data path except that the flow of data occurs in the opposite direction.

Notice that Figure 3.1 shows asynchronous systems producing time-coherent data. The signal processing on the host computer, the USB transfer, and the USRP signal processing all occur at different rates, and without synchronization.\(^3\) Yet, both the transmit and receive data streams remain time-coherent. This is due to the nature of first-in-first-out buffers. FIFOs are a means of interfacing asynchronous system. They provide a place for data producing systems to store data for future retrieval by data consuming systems. A FIFO is said to experience an overflow when it is full at the time a producing system provides data for storage. In such a situation, the FIFO simply ignores the incoming data. An underrun is said to occur when the FIFO is empty at the time a consuming system requests data. In

\(^3\)For instance, the transmit data on the host computer is generated when the appropriate instructions are executed by the host computer’s processor. Since the software is running on a general purpose operating system – as opposed to a real-time operating system – the exact timing of instruction execution cannot be guaranteed or even known. Data is transferred across the USB in much the same way. At the same time, the USRP signal processing paths are operating according to a 64 MHz clock, completely unsynchronized with the host computer.
Figure 3.1: Block diagram of radar software functionality
this situation, the consuming system receives an empty data set.

If the asynchronous systems shown in Figure 3.1 operate such that no overflows or underruns occur at the FIFOs, then the data in the signal processing blocks will be time-coherent. That is, given a sample at index \( n \) that corresponds to some time \( t \), the sample at index \( n + k \) will necessarily correspond to the time \( t + kT \), where \( T \) is the sampling period. Therefore, time-coherence in the transmit and receive data can be maintained by ensuring no overflows or underruns occur at the FIFOs. Note, however, that this does not ensure time-synchronization between the transmitter and receiver. This problem is addressed in the next section.

In order to prevent underruns or overflows, the Tx and Rx signal processing blocks must produce or consume the amount of data requested by the GR framework each time the block is called. Therefore, the transmitter and receiver must be enabled at all times. Because the system is transmitting and receiving simultaneously, each signal path can be allocated one-half of the USB bandwidth. Note that one might be tempted to implement a scheme in which the system is in either transmit or receive mode at a given time. However, this is not possible due to the asynchronous nature of the underlying systems.

### 3.2.2 Transmitter and Receiver Synchronization

If time-coherence is maintained, time-synchronization between the transmit and receive paths can be established so that the time between pulse transmission and reception can be determined. To do this, a loopback is created by terminating the daughterboard transmit and receive ports with 50\(\Omega\) loads, and transmitting a known synchronization signal immediately followed by the radar waveform. See Figure 3.2. There is enough leakage between the transmit and receive lines on the daughterboard to read the signal. The receiver block looks for the known synchronization signal, and after having found it in the receive data stream, can identify the index of the first sample of the leading pulse. At this point, time values relative to the first sample of the leading pulse can be assigned to all incoming samples.
Figure 3.2: Block diagram of software-defined radar in time synchronization calibration mode

Figure 3.3: Block diagram of software-defined radar in operational mode
Note that in the calibration configuration of Figure 3.2, the delay from the output of the FPGA Tx FIFO to the input of the FPGA Rx FIFO should be constant. This is because all operations along this path are synchronous. However, the path from data generation in the software to the input of the FPGA Tx FIFO occurs in software running on a general purpose operating system. As such, due to instruction scheduling variability in the operating system, the delay from the time the transmit block places the first data sample into the Tx FIFO to the time the receive block receives the sample will change each time the application is run. Therefore, this calibration must be performed for each run.

### 3.2.3 Software Architecture Overview

The previous sections addressed the problems of time-coherence and time-synchronization. In this section, a more detailed overview of the software architecture and operation is presented. The reader is urged to refer to Figures 3.1-Figure 3.3 for clarification throughout the discussion. Note that this software is developmental, and is subject to change. Also, Recall that under the GNU Radio framework, signal processing is performed in C++ blocks that are glued together in Python. The Python script also configures the USRP. Only the C++ blocks will be described in the following text. The interested reader is encouraged to examine the source code for more detail.

**The Transmit Signal Processing Block**

A GNU Radio signal processing block was written in C++ to perform the transmit functionality of the radar. The constructor of this block accepts three arguments, each of which are specified by the user. The first argument specifies the file that contains exactly one pulse-repetition interval (PRI) of the radar waveform at baseband. The second argument is the number of times the data in this file should be transmitted (i.e., the number of pulses to transmit). The final argument is the delay, specified in number of samples, that the transmitter should wait before transmitting anything.
Upon instantiation, the transmit block reads and stores the transmit data file to main memory. When the `work()` method of the transmit block is called by the GR framework, the transmitter checks its state and responds accordingly. The system states are: WAIT, SYNC, PREAMBLE, and TRANSMIT. The system starts in the WAIT state. In this state, the transmitter simply counts samples and transmits nothing. Once the requisite number of samples have passed, the system transitions to the SYNC state. In the SYNC state, the system transmits the synchronization signal, which is currently a 5-chip Barker sequence using five samples per chip. After the last sample of the synchronization signal has been placed in the software FIFO, the system transitions to the PREAMBLE state. In the PREAMBLE state the system transmits a known signal. Currently, this signal consists of 1000 zeros, followed by 8000 ones, followed by 1000 zeros. This signal can be used in post-processing to determine the beat frequency between the transmitter and receiver local oscillators on the daughterboard. Once the preamble signal has been transmitted, the system transitions to the TRANSMIT state, where it will transmit the radar waveform data the specified number of times. Figure 3.4 shows a block diagram of the transmit block algorithm.

**The Receive Signal Processing Block**

A GNU Radio signal processing block was written in C++ to perform the receive functionality of the radar. The constructor of this block accepts four arguments, each of which are specified by the user. The first argument is a pointer to the transmit block. This pointer allows the two blocks to communicate. The second argument is the file to which the received data should be stored. The third argument specifies how many samples of each PRI should be recorded to file. The final argument specifies the number of samples to be ignored in each PRI until the receiver should begin recording the number of samples specified by the third argument.

When the `work()` method of the receive block is called by the GR framework, the re-

---

4Note: This architecture could possibly be changed to make use of the `gr.message` and `gr.msg_queue` GNU Radio primitives.
Figure 3.4: Block diagram of transmitter software algorithm
ceiver checks the system state and responds accordingly. If the system is in the WAIT state, the receiver simply ignores all input samples. If the system is in the SYNC, PREAMBLE or TRANSMIT state, this means the transmitter has begun transmitting. In this case, the receiver begins by searching the incoming samples (using a matched filter) for the synchronization signal. If the signal is not found in a specified number of samples, the receiver reports failure, and the application is terminated. If the synchronization signal is found, the receiver begins recording the subsequent samples corresponding to the preamble signal. After the preamble signal has been recorded, the receiver begins recording the specified samples from each PRI. Figure 3.5 shows a block diagram of the transmit block algorithm.

3.3 Hardware

The daughterboards currently offered by Ettus Research are listed in Table 2.1. There are five transceiver models (the RFX* models), and two transmitter/receiver pairs (the Basic* and LF* models). The Basic* and LF* models were eliminated from selection because they do not offer any on-board amplification or filtering. At the time of the selection, only the RFX400, and RFX2400, were available from Ettus, and the RFX2400 was selected because one was readily available. Another RFX2400 and two RFX400 daughterboards were later purchased by the Air Force Research Laboratories (AFRL) for use in this project. One of the RFX2400 boards was modified for coherent operation by following the instructions in [15].

A revision 3 USRP owned by WSU was used for laboratory testing. However an additional USRP revision 4 was purchased by AFRL for use in the project. No modifications to the USRP, including the FPGA firmware, were made.
Figure 3.5: Block diagram of receiver software algorithm
3.4 Hardware Transfer Function

3.4.1 Motivation

Radar Range Resolution

The degree to which a radar system can resolve two targets separated in range is directly proportional to the bandwidth of the radar waveform incident on the target. [14, pp.319] That is, given a waveform bandwidth of $B$, two targets can be resolved by the radar if they are separated in slant range by more than

$$\Delta r = \frac{c}{2\beta}$$

where $c$ is the speed of light. [21, pp. 5] Therefore, an unmodulated short pulse will have better range resolution properties than a long pulse due to the inverse relationship between time and frequency. [3] However, short pulses cannot necessarily be used in all radar applications. One reason for this is that short pulses require greater peak signal power than do longer pulses to maintain a particular signal-to-noise ratio (SNR). Another factor limiting the use of short pulses in radar waveforms is that the radar system circuitry may limit the maximum instantaneous bandwidth of the radar waveform.

Pulse Compression

A signal processing technique known as pulse compression can be employed to circumvent the difficulties associated with the use of short pulses. Pulse compression involves the transmission of a long coded pulse and the processing of the received echo to obtain a relatively narrow pulse. [22, pp. 10.1] In the receiver, pulse compression is implemented by correlating the received signal with a replica of the transmit signal. This is known as matched filtering, which has the additional benefit of maximizing peak-signal-to-noise (power) ratio (SNR) in the presence of additive white Gaussian noise (AWGN). [14, pp.
Waveforms used in pulse compression are chosen primarily on the basis of their autocorrelation function. Desirable autocorrelation functions have a maximum at zero delay, and are nearly zero for all other delays. Examples of pulse compression waveforms are linear frequency modulated (LFM) waveforms and Barker coded waveforms.

Radar System Effects on Pulse Compression

Consider a noiseless radar system ranging a simple point scatter. The transmit waveform will travel out through the transmitter, impinge on the target, and make its way through the receiver to the pulse compression circuitry. Assuming the radar system is linear and time invariant, the entire path from signal generation until just before pulse compression can be modeled by an impulse response $h(t)$. And, the expression for the signal just before pulse compression is given by

$$r(t) = s(t) \ast h(t)$$

where $s(t)$ is the transmit signal and $\ast$ represents the convolution operator. In this scenario, the output of the matched filter will be

$$z(t) = s^*(-t) \ast r(t)$$

$$= s^*(-t) \ast [s(t) \ast h(t)]$$

$$= [s^*(-t) \ast s(t)] \ast h(t)$$ (3.1)

Notice that the bracketed term in Equation (3.1) is the autocorrelation of the transmit signal $s(t)$. As mentioned earlier, the transmit signal is chosen primarily for its autocorrelation properties. However, in this scenario, the system response $h(t)$ also influences the matched filter response, and therefore the pulse compression properties of the waveform cannot be guaranteed.

---

5Recall that the correlation between two signals $a(t)$ and $b(t)$ is equal to the convolution of $a(t)$ with the complex conjugate of $b(t)$ reversed in time. i.e., $a(t) \ast b^*(-t)$. [22, pp. 10.3] Also, since convolution is commutative, the correlation of $a(t)$ and $b(t)$ is also equal to $b(t) \ast a^*(-t)$. [3, pp. 121]
Undoing the Radar System Effects

If the system impulse response \( h(t) \) was known, its inverse filter \( \bar{h}(t) \) could be designed such that

\[
\bar{h}(t) \ast h(t) = \delta(t - \tau)
\]

where \( \delta(\cdot) \) is the Dirac delta function. Once \( \bar{h}(t) \) is known, the transmit signal can be pre-filtered by this inverse filter, and the output of the matched filter would become

\[
z(t) = s^*(-t) \ast r(t)
\]

\[
= s^*(-t) \ast \left[ s(t) \ast \bar{h}(t) \right] \ast h(t)
\]

\[
= s^*(-t) \ast s(t) \ast \left[ \bar{h}(t) \ast h(t) \right]
\]

\[
= s^*(-t) \ast s(t) \ast \delta(t - \tau)
\]

\[
= s^*(-t) \ast s(t - \tau)
\]

which is the desired autocorrelation of the transmit signal delayed by some known \( \tau \). The radar designer can now be satisfied that pulse compression will occur.

### 3.4.2 The USRP-RFX2400 System Model

As we have seen, knowledge of the radar system’s impulse response can be used to improve range resolution. However, this assumes that the system is linear and time-invariant. In this section we attempt to find an expression for the impulse response of the system comprised of the USRP and RFX2400 in order to verify that the system is in fact linear and time-invariant.

We begin by examining the transmit and receive signal paths shown in Figures 2.4, 2.7, and 2.8. By abstracting these figures, we arrive at Figure 3.6, which shows a system level
block diagram of the radar waveform signal path. This figure was derived by noting the following:

Figure 3.6: Block diagram of the USRP transmit and receive signal paths

1. The complex-valued baseband signal $s(t)$ is sent from the host computer to the USRP. This signal is interpolated using CIC and half-band filters, the effects of which are modeled as $h_{t1}(t)$.

2. After interpolation, the signal is digitally up-converted to a frequency of $\omega_{t1}$.

3. The up-converted digital signal is then converted to analog, and is sent to the quadrature modulator on the RFX2400. The effects of the system from interpolator output to modulator input are modeled as $h_{t2}(t)$.

4. The quadrature modulator shifts the signal in frequency from $\omega_{t1}$ by an amount equal to $\omega_{t2}$ while converting from a complex-valued signal to a real-valued signal. A phase offset of $\phi_{t2}$ is also applied.
5. The effect of the system on the signal from the output of the quadrature modulator until just after transmission by the antenna (i.e., upon entering free-space) is modeled by \( h_{23}(t) \).

6. The radiated signal interacts with the environment. This is modeled by \( c(t) \).

7. A portion of the resulting signal returns to the radar, and enters the receive antenna. The effect of the system on the signal from just before reception by the antenna until the reaching the input of the quadrature demodulator is modeled by \( h_{r3}(t) \).

8. The received signal is translated in frequency by an amount equal to \( \omega_r2 \) and a phase offset of \( \phi_r2 \) is applied. The resulting signal is then transformed from real-valued to complex-valued.

9. The demodulated signal is then sent from the RFX2400, to the ADC, and then to the digital down-converters on the FPGA. The effect of the system from the output of the demodulator to the input of the digital down-conversion stage is modeled as \( h_{r2}(t) \).

10. The signal is shifted in frequency by an amount equal to \( \omega_r1 \) and a phase offset of \( \phi_r1 \) is applied.

11. The resulting baseband signal is then decimated using CIC and half-band filters. The effect of the system in this stage is modeled as \( h_{r1}(t) \). The resulting complex-valued baseband signal is sent over the USB to the host computer.

Figure 3.7 shows the system of Figure 3.6 transformed into the frequency domain. Here \( X(\omega) \) represents the Fourier transform of \( x(t) \). The mapping of the system from time to frequency domain is easily performed using the frequency shifting and convolution properties of the Fourier transform. [3, pp. 265]
Figure 3.7: Block diagram of the USRP transmit and receive signal paths represented in the Fourier domain
3.4.3 Finding the Transfer Function

With the system now modeled, we can find an expression of the form

\[ r(t) = s(t) * h(t) \]  

(3.2)

that relates the received waveform \( r(t) \) to the transmit waveform \( s(t) \) via the system impulse response \( h(t) \). In the frequency domain, this become

\[ R(\omega) = S(\omega)H(\omega) \]  

(3.3)

We find an expression like Equation (3.3) by analyzing the signal at each point in the signal path identified by a circled number in Figure 3.7. The following equations correspond to those points.

1.) \( S_1(\omega) = S(\omega)H_{t1}(\omega) \)
2.) \( S_2(\omega) = S(\omega + \omega_{t1})H_{t1}(\omega + \omega_{t1}) \)
3.) \( S_3(\omega) = S(\omega + \omega_{t1})H_{t1}(\omega + \omega_{t1})H_{t2}(\omega) \)

Let \( \omega_t = \omega_{t1} + \omega_{t2} \)

4.) \( S_4(\omega) = e^{j\phi_2}S(\omega + \omega_t)H_{t1}(\omega + \omega_t)H_{t2}(\omega + \omega_{t2}) \)
Combine the system effects modeled as $H_{t3}(\omega)$, $C(\omega)$, and $H_{r3}(\omega)$ into a single term $C_e(\omega)$.

5.) $S_5(\omega) = e^{j\phi_1}S(\omega + \omega_1)H_{t1}(\omega + \omega_1)H_{t2}(\omega + \omega_2)C_e(\omega)$

6.) $S_6(\omega) = e^{j(\phi_1 + \phi_2)}S(\omega + \omega_1 - \omega_{r_2})H_{t1}(\omega + \omega_1 - \omega_{r_2})H_{t2}(\omega + \omega_2 - \omega_{r_2})$

\hspace{1cm} $\times C_e(\omega - \omega_{r_2})$

7.) $S_7(\omega) = e^{j(\phi_1 + \phi_2)}S(\omega + \omega_1 - \omega_{r_2})H_{t1}(\omega + \omega_1 - \omega_{r_2})H_{t2}(\omega + \omega_2 - \omega_{r_2})$

\hspace{1cm} $\times C_e(\omega - \omega_{r_2})H_{r_2}(\omega)$

Let $\omega_r = \omega_{r_1} + \omega_{r_2}$

8.) $S_8(\omega) = e^{j(\phi_1 + \phi_2 + \phi_{r_1})}S(\omega + \omega_1 - \omega_r)H_{t1}(\omega + \omega_1 - \omega_r)H_{t2}(\omega + \omega_2 - \omega_r)$

\hspace{1cm} $\times C_e(\omega - \omega_{r_1})H_{r_2}(\omega - \omega_{r_1})$

9.) $S_9(\omega) = e^{j(\phi_1 + \phi_2 + \phi_{r_1})}S(\omega + \omega_1 - \omega_r)H_{t1}(\omega + \omega_1 - \omega_r)H_{t2}(\omega + \omega_2 - \omega_r)$

\hspace{1cm} $\times C_e(\omega - \omega_{r_1})H_{r_2}(\omega - \omega_{r_1})H_{r_1}(\omega)$

Let $\omega_b = \omega_t - \omega_r$ and $\phi = \phi_{r_2} + \phi_{r_2} + \phi_{r_1}$

$R(\omega) = e^{j\phi}S(\omega + \omega_b)H_{t1}(\omega + \omega_b)H_{t2}(\omega + \omega_b - \omega_{t1})$

\hspace{1cm} $\times C_e(\omega - \omega_{r_1})H_{r_2}(\omega - \omega_{r_1})H_{r_1}(\omega)$ \hspace{1cm} (3.4)

Frequency shifting the received signal by $-\omega_b$ results in

$R(\omega - \omega_b) = S(\omega) \left[ H_{t1}(\omega)H_{t2}(\omega - \omega_{t1}) \right.$

\hspace{2cm} $\times C_e(\omega - \omega_{t})H_{r_2}(\omega - \omega_{t} + \omega_{r_2})H_{r_1}(\omega)e^{j\phi} \left] \right) \hspace{1cm} (3.5)$
Letting $\hat{R}(\omega) = R(\omega - \omega_b)$, and representing the bracketed term in Equation (3.5) by $H(\omega)$, we arrive at an equation similar in form to Equation (3.3)

$$\hat{R}(\omega) = S(\omega)H(\omega)$$

### 3.4.4 Accounting for Oscillator Drift

The relationship of Equation (3.5) assumes that the frequency of each local oscillator is constant in time. However, in reality, oscillators drift, and the system response $H(\omega)$ cannot necessarily be assumed to be time-invariant. We will now reclaim our time-invariant assumption by further examining our system. Before we do so, let us account for the drift by re-writing the translation frequencies as functions of time. We write $\omega_x(t) = \omega_x + \delta_x(t)$ where $x \in \{t1, t2, r1, r2\}$. Here, $\omega_x$ is the commanded frequency and $\delta_x(t)$ is the time-varying frequency error.

The frequency shift by $\omega_{t1}(t)$ in the DDC is done in software. So, this frequency does not drift with time. Similarly, the DUC frequency shift of $\omega_{t1}(t)$ is synthesized digitally by an NCO. It too remains constant over time. Therefore, we can write $\delta_{t1}(t) = \delta_{t1}(t) = 0$. Furthermore, in the standard RFX2400 configuration, $\omega_{t2}(t)$ and $\omega_{r2}(t)$ are synthesized from separate phase-locked loops (PLLs).\(^6\) They therefore drift differently in time. However, as mentioned in Section 2.1.5, a modification to the RFX2400 can be made so that the daughterboard transmit and receive LOs are both driven by the FPGA clock output. In this configuration, $\omega_{t2}(t)$ and $\omega_{r2}(t)$ are synthesized by the same clock, and should therefore experience the same drift. That is $\delta_{t2}(t) = \delta_{r2}(t)$. We will call this drift $\delta_2(t)$. Also, we should note that in operation $\omega_{t1} = \omega_{r1}$ and $\omega_{t2} = \omega_{r2}$.

We can now use the above system information to re-write Equation (3.5). First, we

\(^6\)Model number ADF4360 by Analog Devices
examine the beat frequency between the transmitter and receiver, and find that

\[ \omega_b = \omega_t(t) - \omega_r(t) \]

\[ = \omega_{t1}(t) + \omega_{t2}(t) - \omega_{r1}(t) - \omega_{r2}(t) \]

\[ = (\omega_{t1} + \delta_{t1}(t)) + (\omega_{t2} + \delta_{t2}(t)) - (\omega_{r1} + \delta_{r1}(t)) - (\omega_{r2} + \delta_{r2}(t)) \]

\[ = \omega_{t1} + \omega_{t2} - \omega_{r1} - \omega_{r2} \]

\[ = 0 \]

Plugging this into Equation (3.5), and simplifying we find

\[ R(\omega) = S(\omega) \left[ H_{t1}(\omega) H_{t2}(\omega - \omega_{t1}) \times C_e(\omega - \omega_{t1} - \omega_{t2} - \delta_{2}(t)) H_{r2}(\omega - \omega_{t1}) H_{r1}(\omega)e^{j\phi} \right] \quad (3.6) \]

In this relationship, only \( \delta_{2}(t) \) varies with time. If we assume that the FPGA clock drift is negligible, we can write the following linear, time-invariant relationship between the transmit and receive signals.

\[ R(\omega) = S(\omega) \left[ H_{t1}(\omega) H_{t2}(\omega - \omega_{t1}) \times C_e(\omega - \omega_{t1} - \omega_{t2}) H_{r2}(\omega - \omega_{t1}) H_{r1}(\omega)e^{j\phi} \right] \]

3.4.5 Conclusion

As expected, the true system is linear, but not time-invariant. However, it can be closely approximated by a linear time-invariant system. One should note that the phase term \( \phi \) does not vary with time after initial power up. However, it will vary between USRP power cycles. This unknown \( \phi \) in the above equation will merely rotate the signal in phase by a constant amount, and this should not significantly effect system performance.
Chapter 4

System Tests

The software-defined radar (SDR) discussed in Chapter 3 must be capable of performing each of the following tasks:

1. Generating the desired radar waveform in software
2. Passing the generated waveform from software to hardware
3. Transmitting the generated waveform
4. Receiving the return signal
5. Passing the return signal from hardware to software
6. Recording the desired portions of the return signal in software

The ability of the prototype SDR to perform each of these tasks was tested in the laboratory, and the results of these tests are presented in this chapter. Throughout the discussion of the test results, the reader may find it helpful to refer to Figure 4.1, which shows a conceptual block diagram of the SDR architecture in which portions of the SDR corresponding to each of the six tasks have been labeled. This figure illustrates the components relevant to each task, and indicates which components must be isolated during a particular test.
4.1 Waveform Generation and Recording

Radar waveform generation and recording, tasks 1 and 6 respectively, are purely software functions. During waveform generation, the desired waveform is generated by the Tx block and passed into a software FIFO. Conversely, during waveform recording, the Rx block reads data from a software FIFO, and records it to a file. Hence, tasks 1 and 6 were tested in the lab with the hardware (USRP and RFX2400) “out of the loop.” This test configuration is illustrated in Figure 4.2.

To test waveform generation and recording, one period of a sinusoid was loaded into the Tx block, and transmitted repeatedly into the software Tx FIFO. The Rx block was configured to receive only samples corresponding to particular times (i.e., ranges). The results of these tests are shown in Figures 4.3-4.5. The top plot in each figure shows exactly one period of the sinusoid. The solid red triangles in the plot represent the samples of each period that the Rx block was configured to record. The bottom plot in each figure shows the actual waveform recorded by the Rx block. It can be observed that in each test, the
Figure 4.2: SDR configuration during software waveform generation and recording tests

bottom plot contains only the desired samples. Since the software operates independently of the USRP, it should be expected that the radar transmit and receive functions will operate correctly when the hardware is in the loop. This is in fact the case, and verification of radar transmission and reception with the hardware in the loop is demonstrated via the system transfer function compensation tests in Section 4.3.
Figure 4.3: Ranging test using a delay of 5 samples and window of 11 samples

Figure 4.4: Ranging test using a delay of 21 samples and window of 11 samples
Figure 4.5: Ranging test using a delay of 32 samples and window of 11 samples
4.2 Transmission

The passing of waveform data from software to hardware (task 2) and the transmission of this waveform by the RFX2400 (task 3) were tested by injecting RFX2400 output into a spectrum analyzer (SA), and comparing the actual spectrum with expected results.\textsuperscript{1} Ideally, the waveforms would be compared in time as well, but equipment capable of performing these tests could not be obtained. The test setup is shown in Figure 4.6.

Table 4.1 lists the waveforms used in the tests, and Figures 4.7-4.9 show their expected spectra. These figures were generated by filtering the Barker signals by the two-way system impulse response, and then taking the Fourier transform. The two-way (transmit and receive) system impulse response is discussed in detail in Section 4.3. However, it should be mentioned here that the one-way, transmit-only system impulse response should have been used, but it could not be measured directly. Therefore, the two-way system response was used as an approximation. Also, the spectral lines beneath the envelope are caused by the pulse repetition. The pulses repeat every $T_p = 12.5\mu s$. Therefore, the spectral lines are separated by $f_p = 1/T_p = 80$ kHz. Figures 4.10-4.12 show the actual spectra as measured by the spectrum analyzer.

<table>
<thead>
<tr>
<th>ID</th>
<th>Center Freq. (GHz)</th>
<th>PRI ($\mu s$)</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>2.35</td>
<td>12.5</td>
<td>5-chip Barker, 1 sample/chip</td>
</tr>
<tr>
<td>S2</td>
<td>2.35</td>
<td>12.5</td>
<td>5-chip Barker, 2 sample/chip</td>
</tr>
<tr>
<td>S4</td>
<td>2.35</td>
<td>12.5</td>
<td>5-chip Barker, 4 sample/chip</td>
</tr>
</tbody>
</table>

Table 4.1: Waveforms used to test radar transmission

\textsuperscript{1}The spectrum analyzer used was an Anritsu Spectrum Master, model number MS2721A, on loan from AFRL/SNAT.
Figure 4.6: Hardware configuration for spectral measurements

Figure 4.7: Expected power spectrum of a 5-chip Barker, 1 samples/chip, 12.5 microsec. PRI
Figure 4.8: Expected power spectrum of a 5-chip Barker, 2 samples/chip, 12.5 microsec. PRI

Figure 4.9: Expected power spectrum of a 5-chip Barker, 4 samples/chip, 12.5 microsec. PRI
Figure 4.10: Actual power spectrum of a 5-chip Barker, 1 samples/chip, 12.5 microsec. PRI

Figure 4.11: Actual power spectrum of a 5-chip Barker, 2 samples/chip, 12.5 microsec. PRI
Figure 4.12: Actual power spectrum of a 5-chip Barker, 4 samples/chip, 12.5 microsec. PRI
4.3 System Transfer Function Compensation

4.3.1 Motivation

The ideal matched filter response for the length-5 Barker sequence \{1, 1, 1, -1, 1\} is shown in Figures 4.13 and 4.14 for 1 sample per chip and 2 samples per chip, respectively. One can observe that in this ideal case, the length-5 Barker sequence experiences a gain of five in peak sidelobe ratio as well as a range compression factor of five. It is desirable for the prototype SDR to experience pulse compression performance as close to this ideal performance as possible.

![Normalized Pulse Compression Response](image)

Figure 4.13: Ideal matched filter response of a length-5 Barker sequence comprised of 1 samples per chip.

---

Note that the matched filter input and output are normalized such that their respective peak amplitudes are unity. Scaling the signals in this manner helps one to compare the true gain in pulse compression for both peak sidelobe ratio and time compression.

---

2Note that the matched filter input and output are normalized such that their respective peak amplitudes are unity. Scaling the signals in this manner helps one to compare the true gain in pulse compression for both peak sidelobe ratio and time compression.
4.3.2 Test Setup

Ideally, to test pulse compression in our prototype SDR, we would transmit and receive our signal via a non-dispersive RF delay line of sufficient length and known frequency response. However, a delay line of sufficient length could not be obtained. Therefore, to measure the USRP/RFX2400 impulse response, the system was configured as shown in Figure 4.1, and a single sample (i.e., impulse) was transmitted through the system.\(^3\) We call this configuration the zero-delay configuration because there is zero delay between the Tx and Rx ports as compared to transmission through an RF delay line. It is expected that the system characteristics in the actual operational configuration will differ slightly from those of the system in this zero-delay configuration. However, we do not anticipate these differences to be significant. Also, as will be discussed in the next chapter, an operational system will need to use the Rx\(^2\) port of the RFX2400, as opposed to the Tx/Rx port. Therefore, all of the operational tests were performed with the Rx\(^2\) port as the receiving port.

\(^3\)Actually, this test was repeated many times, and the results were averaged together to form an estimate of the impulse response.
4.3.3 Test Results

Figure 4.15 shows the actual matched filter response of the USRP + RFX2400 in which the RFX2400 has been modified for coherent operation. Evidently, system effects cause the matched filter response to deviate slightly from the ideal. This is to be expected when only one sample-per-chip is used in the transmission of the Barker sequence. The ADCs and DACs have a bandwidth of $f_s$, where $f_s$ is the sampling frequency, which is also the bandwidth of one sample. Therefore, for any system, no matter how fast the converters are clocked, one should expect to see distortion in the transmit signal if only one sample per chip is used.

![Normalized Pulse Compression Response](image)

Figure 4.15: Actual matched filter response of a length-5 Barker sequence (1 samples/chip) with no pre-filtering to remove system effects

A corollary to the above statement is that signal distortion due to system effects should lessen if the Barker signal is composed of two or more samples per chip. This was tested by transmitting the same length-5 Barker sequence as before, only this time with two samples per chip. The results of these tests is shown in Figure 4.16. As expected, signal distortion

---

4The pulse compression performance when the Tx/Rx port is used to receive is actually worse than when the Rx2 port is used. However, those results will not be shown here.
diminishes as the number of samples per chip increases. However, distortion due to system effects is still present. As we will see in Chapter 5, we may need to use one sample per chip in order to maximize instantaneous signal bandwidth. Therefore, we desire a means of reducing system distortion for the one sample per chip signal. This is the topic of the next section.

![Figure 4.16: Actual matched filter response of a length-5 Barker sequence (2 samples/chip) with no pre-filtering to remove system effects](image)

**4.3.4 Zero-Delay Transfer Function**

If the impulse response of a system is known, then an inverse filter can be constructed and used to pre-filter the transmit waveform. If the system is linear and time-invariant, then pre-filtering the transmit waveform should have the effect of removing system distortions from the received waveform. To that end, the system’s impulse response was measured, and an inverse filter constructed. The results of this work are presented here.

Figure 4.17 shows a smoothed version of the measured frequency response of the system, and Figure 4.18 shows the corresponding impulse response. Notice that the system
is basically a low-pass filter. The frequency response is shown over 4 MHz, which is the sampling frequency required when operating in simultaneous transmit/receive mode with 4 bytes per sample. However, the system appears to only have a -3 dB bandwidth of 1.5 MHz.

The inverse filter shown in Figure 4.19 was constructed by taking the inverse of the system frequency response, and Figure 4.20 shows that the filter of Figure 4.19 really is the inverse of the response shown in Figure 4.17. The top plot in Figure 4.20 shows the Barker sequence to be transmitted, and the middle plot shows the signal pre-filtered by the system inverse filter. The bottom plot shows this pre-filtered signal filtered by the system impulse response. As expected, the original signal is perfectly reconstructed. Note that this simulation does not imply the true system inverse filter has been obtained. Instead, this simulation implies that the inverse filter obtained is the true inverse of the measured impulse response, which is itself an average of a number of realizations.

![Normalized frequency response](image)

**Figure 4.17: Normalized frequency response**

Figure 4.21 shows the results when the length-5 Barker signal was pre-filtered by the inverse filter, transmitted, received, and match filtered with the coherently configured
Figure 4.18: Impulse response

Figure 4.19: Normalized frequency response inverse filter
Figure 4.20: Simulated effect of pre-filtering transmit waveform with inverse filter

RFX2400 in the zero-delay configuration. For the 1 sample per chip case, pre-filtering slightly improves the matched filter response compared to the uncompensated signal in Figure 4.15. This improvement is less than expected, but could be the result of inaccuracies in system frequency response measurement. For a two chip per sample signal, pre-filtering has almost no effect. This was expected because the system has less distortion for more narrow band signals. This result is shown in Figure 4.22.
Figure 4.21: Actual matched filter response of a length-5 Barker sequence (1 samples/chip) when pre-filtering has been applied to remove system effects

Figure 4.22: Actual matched filter response of a length-5 Barker sequence (2 samples/chip) when pre-filtering has been applied to remove system effects
4.4 Phase Coherence

Another very important receiver characteristic is phase coherence, which refers to the difference in phase between the transmit and receive signals. The coherently configured system was placed in the zero-delay configuration, and a pulsed Barker-coded signal was transmitted and received. Figures 4.23-4.26 show the resultant cross-spectral analysis plots. Figures 4.23 and 4.24 correspond to no pre-filtering before transmission and Figures 4.25 and 4.26 correspond to the case when the signal was pre-filtered before transmission. In both cases, the one can see that the system is moderately coherent.

Figure 4.23: Phase coherence plot of received signal with no pre-filtering to remove system effects
Figure 4.24: Phase coherence plot of received signal with no pre-filtering to remove system effects (Close-up of Figure 4.23)

Figure 4.25: Phase coherence plot of received signal with pre-filtering to remove system effects
Figure 4.26: Phase coherence plot of received signal with pre-filtering to remove system effects (Close-up of Figure 4.25)
4.5 Further Testing

The purpose of this research has been to develop a proof of concept software-defined radar. As such, only a few key radar properties need to be measured and verified. These are: time coherence and phase coherence – that is, the ability of the radar to transmit and receive radar waveforms without dropping samples or overly distorting the signal. This capability has been demonstrated here. In order to determine exactly what type of radar this prototype can become, one may be interested in more refined tests to include:

- Analyzing transmissions in the time domain, not just the spectral domain.
- Free-space tests in which the system radiates into free space, and the returns are analyzed.
- Maximum transmit power – increase transmit power until we start to see nonlinearities and third order products.
- Minimum detectable signal (i.e., receiver sensitivity.) – Receiver sensitivity is a difficult parameter to measure because in a radar context it denotes the minimum received signal power that can be detected, and in order to determine this value, the radar waveform must be specified. To test MDS, we suggest the test configuration shown in Figure 4.27. This configuration is comprised primarily of two RFX2400’s, driven by the same clock, such that one RFX2400 transmits a waveform of known signal power into the other RFX2400.
Figure 4.27: Hardware configuration for radar simulator.
Chapter 5

Predicted Performance

In this chapter, we discuss the expected performance of the GNU Radio and USRP based software-defined radar in an operational context. This assessment is based upon knowledge of the system (Chapter 2) and system tests (Chapter 4). The result of this analysis is that, in general, a GNU Radio and USRP based SDR will most likely not be able to meet high bandwidth application requirements. However, some inherent performance limitations may be surmountable with software and hardware extensions and modifications.

5.1 Down-Range Resolution

The USRP is configured to provided 4 bytes-per-sample (16 bits real, 16 bits imaginary) in both transmit and receive mode, and the USB interface chip on-board the USRP can stream at a maximum rate of 32 MBps (megabytes-per-second). With the SDR in full-duplex mode (i.e., always transmitting and receiving), which is required to maintain time-coherence, the USB interface becomes the bottleneck allowing only 4 MHz of bandwidth.\(^1\) A 4 MHz Barker signal was successfully processed during the system tests. Therefore we know that signals of this bandwidth can be processed by the system. As a result, we can expect a

\[^1\]32 \text{MBps}/(4\text{B/S} + 4\text{B/S}) = 4\text{S/s} = 4 \text{MHz}
minimum range resolution of 37.5 meters.\(^2\)

This range resolution of 37.5 meters will most likely be unacceptable for many applications (e.g., GMTI and SAR). Hence, resolution must be improved. An analysis of the system architecture (Chapter 2) reveals that an increase in signal bandwidth will require system modifications, which will likely occur in one of three places:

**Fewer Bits-Per-Sample:** If a radar application does not require 32 bits-per-sample (b/S), then modifying the FPGA firmware to require only \(N_b\) b/S would increase the instantaneous bandwidth of the system. The formula for calculating \(N_b\) as a function of instantaneous bandwidth is given as

\[
N_b = 32 \cdot 10^6 \frac{\text{bytes}}{\text{second}} \times 8 \frac{\text{bits}}{\text{byte}} \times 1 \frac{1}{\text{BWseconds}} \times \frac{1}{2}
\]

Note that \(N_b\) must be an even integer number. Once \(N_b\) is determined, dividing it by 4 gives the number of bits-per-channel-per-component (i.e., real or imaginary component) required. The maximum attainable bandwidth using this method is 64 MHz, which results in 1 bit-per-channel-per-component.

**Stepped Frequency:** The FPGA firmware could be modified to step the center frequency of the waveform to cover some desirable bandwidth within one pulse repetition interval (PRI). The data sheet for the ADF4360 phase-locked loop (PLL), which is used to set the carrier frequency, lists a typical frequency lock time of 250\(\mu\)s. Therefore, the radar’s pulse-repetition frequency (PRF) will be limited by the settling time of the PLL.

**Hardware Modifications:** The USB 2.0 interface between the USRP and the host PC is the bandwidth bottleneck. Bandwidth could be increased if the USRP were modified to have a faster interface like PCI or Gigabit Ethernet. However, the daughterboards would

\(^2\text{range resolution} = \text{speed of light} / (2 \times \text{bandwidth}) \ [21, \text{p.} \ 5] \)
have to be examined to ensure that they can handle more bandwidth. A cursory examination of the RFX2400 receive chain (See Figure 2.7) reveals a 20 MHz filter that would have to be modified. There are probably many other components that would have to be replaced. In light of the complexity involved with making these hardware modifications, this method is not recommended.

5.2 Power Budget

In order to determine maximum detection ranges and minimum target radar cross-sections, we must first calculate the radar’s power budget. Figure 5.1 shows the RFX2400 components relevant in these power budget calculations. Notice that the box labeled RFX2400 Daughter board contains components on the RFX2400 while the box labeled External Processing contains components that are to be determined by the radar system designer. The circulator and low-noise amplifier (LNA) shown in this box are listed with values typical of readily available COTS components. It is important to note that the external circuitry is required for increased transmit power and transmitter/receiver isolation.

Figure 5.2 shows a plot of radar SNR versus target cross section and range. The standard radar range equation of [14]

\[ R = \frac{P_t G_t G_r \lambda^2 \sigma}{(4\pi)^3 R^4 kTBFL} \]  

(5.1)

was used. The values for each parameter in Equation (5.1) is given in Table 5.1. Note that for a 1 m² target at 1 km slant range, the system is estimated to have a SNR of -21 dBm. Also note that signal processing gain was not included in these calculations because these gains are application specific.
Figure 5.1: Simplified block diagram of USRP transmit/receive system that illustrates key components in power budget calculations.
Table 5.1: Radar range equation parameters

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Value</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_t$</td>
<td>Peak transmit power</td>
<td>0.1</td>
<td>Watts</td>
</tr>
<tr>
<td>$G_t$</td>
<td>Transmit antenna gain</td>
<td>100</td>
<td>unitless</td>
</tr>
<tr>
<td>$G_r$</td>
<td>Receive antenna gain</td>
<td>100</td>
<td>unitless</td>
</tr>
<tr>
<td>$\lambda$</td>
<td>Wavelength</td>
<td>0.125</td>
<td>meters</td>
</tr>
<tr>
<td>$\sigma$</td>
<td>Target radar cross section</td>
<td>–</td>
<td>meters</td>
</tr>
<tr>
<td>$R$</td>
<td>Target slant range</td>
<td>–</td>
<td>meters</td>
</tr>
<tr>
<td>$k$</td>
<td>Boltzmann’s constant</td>
<td>$1.3806503 \cdot 10^{-23}$</td>
<td>$\text{m}^2\text{kg/s}^2\text{K}^{-1}$</td>
</tr>
<tr>
<td>$T$</td>
<td>Temperature</td>
<td>290</td>
<td>Kelvin</td>
</tr>
<tr>
<td>$B$</td>
<td>System bandwidth</td>
<td>$4 \cdot 10^6$</td>
<td>Hz</td>
</tr>
<tr>
<td>$F$</td>
<td>System noise figure</td>
<td>2.82</td>
<td>unitless</td>
</tr>
<tr>
<td>$L$</td>
<td>System losses</td>
<td>4</td>
<td>unitless</td>
</tr>
</tbody>
</table>

Figure 5.2: SNR vs. target range and radar cross section. Max = 17 dB. Min = -21 dB. (Signal processing gain not included.)
5.3 Waveform Selection

Most applications will require maximum bandwidth from the SDR. This maximum bandwidth can only be achieved if signaling is performed using single samples – e.g., a Barker sequence with 1 sample per chip. Furthermore, the isolation between the transmit and receive lines is not great enough to allow continuous waveform (CW) operation. As such, the system must operate in pulsed mode. These requirements of pulsed operation and maximum bandwidth restrict the radar to phase code waveforms such as Barker sequences. Frequency modulation cannot be used because these waveforms require more than one sample to achieve their total bandwidth.

5.3.1 Dead Zone

A radar designer may wish to enhance the SNR of the system by taking advantage of the signal processing gain available to phase coded waveforms. However, as the signal processing gain of these waveforms increases, so does the pulse width. And, as pulse width increases, so does the dead zone. The dead zone is the range from the radar within which all target returns arrive before the pulse is completely transmitted.[14] Targets within this range are therefore eclipsed, and go undetected.

The equation for the dead zone is given by

\[ R_d = \frac{c\tau}{2} \]

where \( \tau \) is the pulse width.[14] The dead zone was calculated for a number of signal processing gains (i.e., number of chips) and bandwidths (i.e., number of samples per chip.) Table 5.2 shows these values in tabular form, while Figure 5.3 shows the same data graphically. This data can be used along with Figure 5.2 to design a radar waveform for a particular stand-off range.
<table>
<thead>
<tr>
<th>Processing Gain</th>
<th>Bandwidth (MHz)</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.80</td>
<td>1.00</td>
<td>1.33</td>
<td>2.00</td>
<td>4.00</td>
</tr>
<tr>
<td>2</td>
<td>0.37</td>
<td>0.30</td>
<td>0.22</td>
<td>0.15</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>0.56</td>
<td>0.45</td>
<td>0.34</td>
<td>0.22</td>
<td>0.11</td>
</tr>
<tr>
<td>4</td>
<td>0.75</td>
<td>0.60</td>
<td>0.45</td>
<td>0.30</td>
<td>0.15</td>
</tr>
<tr>
<td>5</td>
<td>0.94</td>
<td>0.75</td>
<td>0.56</td>
<td>0.37</td>
<td>0.19</td>
</tr>
<tr>
<td>7</td>
<td>1.31</td>
<td>1.05</td>
<td>0.79</td>
<td>0.52</td>
<td>0.26</td>
</tr>
<tr>
<td>11</td>
<td>2.06</td>
<td>1.65</td>
<td>1.24</td>
<td>0.82</td>
<td>0.41</td>
</tr>
<tr>
<td>13</td>
<td>2.44</td>
<td>1.95</td>
<td>1.46</td>
<td>0.97</td>
<td>0.49</td>
</tr>
</tbody>
</table>

Table 5.2: Dead zone (km) vs. signal processing gain and bandwidth. (Max = 2.44 km. Min = 0.07 km.)

Figure 5.3: Dead zone (km) vs. signal processing gain and bandwidth. (Max = 2.44 km. Min = 0.07 km.)
5.4 Looking Toward the Future

The USRP project is perpetually changing. As of this writing (March 2007) the USRP is in revision 4. In a personal correspondence to the author, Ettus indicates that he is currently working on a second generation USRP that will have Gigabit Ethernet instead of USB 2.0 as well as 100 MHz quadrature converters instead of the 64 MHz quadrature converters. Ettus says that this should allow 25 MHz instantaneous bandwidth at 16-bits/sample or 50 MHz at 8-bits/sample. However, some modifications to the daughterboards will probably be required to accommodate this wider bandwidth. These second generation USRPs will also have a more powerful FPGA as well as 1 MB of RAM on-board. This should allow some of the most intensive signal processing to be done on the FPGA, which may open the door for more complicated architectures. Looking even further into the future (1+ years), Ettus hopes to produce a USRP with a PCI express connection. This should provide over 100 MHz bandwidth at 16-bits/sample.

There are changes on the GNU Radio horizon as well. A current effort to re-work the GNU Radio architecture is being undertaken by Eric Blossom and BBN Technologies. This work is aimed at providing GNU Radio with packet-based communication capabilities including in-band signaling and precision timing. These modifications, along with a more powerful FPGA, should result in a platform more amenable to radar applications, and may prove to be a very useful asset to the radar research community.

5.5 Conclusion

The GNU Radio project and USRP hardware can be used to create a software-defined radar. However, the instantaneous bandwidth and transmitter/receiver isolation severely limit the performance of the system. These limitations may be surmountable if the application specifications permit.
Bibliography


Part II

Nonquadratic Regularization for Waveform Optimization
Chapter 6

Nonquadratic Regularization for Waveform Optimization

The low cost software-defined radar prototype of the first part of this thesis will most likely suffer operational constraints on transmit power and computing resources. Therefore, it is desirable to field a system which is capable of optimizing its signal processing based upon the operational environment. In this chapter, an adaptive algorithm for eigen-based waveform optimization is presented. This algorithm is capable of improving the matched filter signal-to-interference-plus-noise ratio (SINR) of a radar operating in a colored interference environment while simultaneously constraining the shape of the matched filter response. To arrive at the algorithm cost function, the asymptotic behavior of the interference covariance matrix is examined, and a $p$-norm regularization term is considered. The resulting cost function consists of one term to improve SINR, and another term to constrain the shape of the matched filter response. This provides the waveform designer with a means of trading between SINR performance and matched filter response for a given application. The simulated performance of various steepest descent algorithms applied to waveforms in a colored interference environment is presented. These results show that a low-complexity stochastic gradient algorithm (in the spirit of LMS) is capable of improving radar performance in
6.1 Introduction

Radar waveform optimization is a technique that can be used to adjust any or all of the radar transmitter degrees of freedom based upon observations of a dynamic interference environment. Changes in the transmit signal then necessitate changes in the receiver processing and matched filter itself. Many proposed solutions to this problem center around the eigen decomposition of the estimated interference plus noise covariance matrix.[23]-[25]

A waveform optimization technique put forth by Bergin et al.[23] uses the projection of a template waveform onto the low-noise subspace of the interference covariance eigen-space to improve matched filter SINR while simultaneously maintaining a desirable matched filter response. In this way, the SINR is improved while desirable pulse-compressed characteristics such as narrow range mainlobe width and low range sidelobe levels are preserved. However, this projection into the low-noise subspace requires the estimation and eigen decomposition of the interference covariance matrix along with other operations that may be too computationally intensive for many applications.

As an alternative, an adaptive algorithm with low computational complexity has been developed that is capable of improving SINR while constraining the shape of the matched filter response. The cost function at the heart of this algorithm consists of two terms: one term to improve SINR, and another term to constrain the matched filter response. Constraint of the matched filter response is generally achieved at the cost of SINR improvement. However, the weighting of each cost function term may be chosen independently of the other. This provides the waveform designer with a means of trading between SINR performance and matched filter response for a given application.

To arrive at the algorithm cost function, an expression of the SINR is developed for an assumed signal model, and the asymptotic behavior of the interference covariance matrix
is examined. The addition of a $p$-norm regularization term to the cost function is then considered. Various steepest descent update equations for the cost function are found, and the simulated performance of each update equation is discussed. It is shown that a low-complexity stochastic gradient algorithm (in the spirit of LMS) is capable of improving radar performance in band-limited interference.

The following notation will be used hereafter. Column vectors are denoted by underlined lowercase letters, and matrices are denoted by uppercase letters. Vector elements are denoted using subscripts (e.g., $x_m$), while time is indexed in parentheses (e.g., $x(n)$). The conjugate transpose of a vector or matrix is denoted by the superscript $(\cdot)^H$, and complex conjugation is denoted by $(\cdot)^*$. The vector element-by-element multiplication operator is represented by $\odot$. The $p$-norm of a vector is denoted by $||x||_p = (|x_1|^p + |x_2|^p + \cdots + |x_N|^p)^{\frac{1}{p}}$, and $\nabla_x$ denotes the gradient operator with respect to the vector $x$. Finally, $I$ is the identity matrix, and $j = \sqrt{-1}$.

### 6.2 Development

#### 6.2.1 Signal Model

Consider a radar system operating in a colored interference environment. Let the transmitted signal be represented in discrete time by the length-$N$ sequence $q(k)$ defined by

$$ q = \begin{bmatrix} q(0) \\ q(1) \\ \vdots \\ q(N-1) \end{bmatrix} \equiv Uw \quad (6.1) $$
where the columns of $U$ are the length-$N$ IDFT basis vectors, and for future use, $w$ is the DFT of $q$.

The received radar signal ($r$) comprises an attenuated, delayed, and Doppler-shifted version of the transmit signal, as well as receiver noise ($n$), and the colored interference signal ($i$) such that

$$r(k) = b e^{-j\omega_d k} q(k - \tau_0) + i(k) + n(k)$$

where $b$ is the attenuation constant, and $\omega_d$ is the Doppler frequency shift. Under the assumption that the receiver noise is spectrally white, we can combine the interference and thermal noise into a single quantity without loss of generality. In addition, we assume that the Doppler shift observed by a single pulse is small relative to the signal bandwidth such that the received signal can be approximated by

$$r(k) \approx b e^{-j\phi_d} q(k - \tau_0) + i(k) \quad (6.2)$$

where $\phi_d$ is the Doppler phase shift of a single pulse.

### 6.2.2 Matched Filter SINR

Given the expression for the received signal in (6.2), the output of the matched filter can be written as

$$m(\tau) = \sum_{k=0}^{N-1} q^*(k - \tau) r(k)$$

$$= \sum_{k=0}^{N-1} q^*(k - \tau) \left( b e^{-j\phi_d} q(k - \tau_0) + i(k) \right)$$
In the absence of interference, the maximum of the matched filtered response should occur at \( \tau_0 \) seconds such that

\[
m(\tau_0) = \sum_{k=0}^{N-1} q^*(k-\tau_0) \left( b e^{-j\phi_d} q(k-\tau_0) + i(k) \right)
\]

\[
= \sum_{k=0}^{N-1} b e^{-j\phi_d} q^*(k-\tau_0) q(k-\tau_0) + q^*(k-\tau_0)i(k)
\]

\[
= \sum_{k=0}^{N-1} b e^{-j\phi_d} q^*(k)q(k) + \sum_{k=0}^{N-1} q^*(k)i(k + \tau_0)
\]

\[
= b e^{-j\phi_d} q^H q + q^H \hat{\mathbf{i}}_{\tau_0}
\]

where \( \hat{\mathbf{i}}_{\tau_0} = [i(\tau_0) \ i(\tau_0 - 1) \ \cdot \cdot \cdot \ i(\tau_0 - N + 1)]^T \).

Hence, the signal-to-interference-plus-noise ratio of the maximum matched filter response is

\[
SINR = \frac{|m_s(\tau_0)|^2}{E\{|m_s(\tau_0)|^2\}}
\]

\[
= \frac{(b e^{-j\phi_d} q^H q)(b e^{-j\phi_d} q^H q)^H}{E\{(q^H \hat{\mathbf{i}}_{\tau_0})(q^H \hat{\mathbf{i}}_{\tau_0})^H\}}
\]

\[
= \frac{b^2 q^H q q^H q}{q^H E\{\hat{\mathbf{i}}_{\tau_0} \hat{\mathbf{i}}_{\tau_0}^H\} q}
\]

\[
= \frac{b^2}{q^H R_i(k + \tau_0) q}
\]

where \( R_i(k + \tau_0) \) is the interference covariance matrix at time \( k + \tau_0 \). Assuming the interference is wide-sense stationary over the sensing interval, \( R_i(k + \tau_0) \) can be replaced with \( R_i \), which is the interference covariance matrix at the beginning of the sensing interval.

Making the simplifying assumptions of unit target reflectivity \( b^2 = 1 \), and a transmitted
signal with power normalized to $||q||_2^2 = 1$, the SINR becomes

$$SINR = \frac{1}{q^H R_i q}$$  \hspace{1cm} (6.3a)$$

As the size of the Toeplitz matrix $R_i$ becomes large, its eigenvectors tend to the discrete Fourier transform basis vectors [26], such that $R_i \approx U \Lambda U^H$ where $U^H$ is the DFT matrix. Therefore, assuming large $N$, we can write the SINR as

$$SINR = \frac{1}{w^H U \Lambda U^H U w}$$  \hspace{1cm} (6.4a)$$

$$= \frac{1}{w^H \Lambda w}$$

$$= \frac{1}{\sum_{m=0}^{N-1} \lambda_m |w_m|^2}$$  \hspace{1cm} (6.4b)$$

The form of the SINR in equations (6.4a), and (6.4b) clearly show that a waveform optimal in an SINR sense is one in which all energy is projected along $u_{\min}$, where $u_{\min}$ is the eigenvector corresponding to $\lambda_{\min}$, the smallest eigenvalue of $R_i$.[23] Due to the Toeplitz nature of $R_i$, each eigenvalue $\lambda_m$ is the power spectral density of the interference at each of the DFT basis frequencies. Therefore, the optimal transmit waveform places all signal energy in the portion of the spectrum least occupied by the interference, which improves its spectral efficiency.

### 6.2.3 Cost Function

Based on the above development, we may maximize SINR by choosing $q$ to minimize $q^H R_i q$. However, for many applications the shape of the matched filter response is critical (e.g., mainlobe width and peak sidelobe level). Therefore, the cost function should be one that penalizes for increased mainlobe width and raised peak sidelobe levels in the matched filter response as well as for low SINR.
Inclusion of a nonquadratic regularization term has been practically demonstrated to emphasize low side lobes and a narrow main lobe width. Applications include SAR image autofocus [27]-[28] and image formation [29]. To optimize the transmitted signal, we choose to minimize the regularized cost function

\[ C(q) = q^H R_i q + \gamma ||q||_p^p \]  

(6.5)

where \( \gamma \) is a constant, and can be selected to control the contribution of the \( p \)-norm regularization term.

### 6.2.4 Gradient Descent Algorithm

Any number of adaptive algorithms can be used to minimize the cost function. If the interference covariance matrix were known \textit{a priori}, a steepest descent algorithm could be employed in which case the update equation is given by

\[ q(t+1) = q(t) - \mu \left[ \nabla_q C(q(t)) \right] \]

where \( \mu \) is a constant called the \textit{step size}.\[30\] In this case, the cost function gradient is

\[ \nabla_q C(q) = \nabla_q (q^H R_i q + \gamma ||q||_p^p) = 2R_i q + \gamma \frac{p}{2} |q|^{p-2} \odot q \]

where \( |q|^{p-2} \) denotes a vector in which the \( i^{th} \) element is equal to \( |q_i|^{p-2} \). Subsequently, the gradient descent update equation becomes

\[ q(t+1) = q(t) - \mu R_i q - \gamma p |q|^{p-2} \odot q \]  

(6.6)
where \( \mu \) absorbs a factor of 2 and \( \gamma \) absorb a factor of \( \frac{\nu}{2} \) without loss of generality. While it may be impractical to assume prior knowledge of \( R_i \) in real-time applications, the gradient descent can provide a baseline for performance comparisons.

### 6.2.5 Stochastic Gradient Descent Algorithm

In the more practical situation where the interference covariance matrix is not known \textit{a priori}, it can be instantaneously approximated by

\[
R_i = E\{i^H i\} \approx i^H i
\]  

(6.7)

or recursively estimated with

\[
\hat{R}_i(t) = R_i(t-1)\alpha + i^H i (1 - \alpha)
\]  

(6.8)

where the forgetting factor \( \alpha \) exists on the interval [0, 1]. In either case, the stochastic estimate of \( R_i \) is then substituted into (6.6). The stochastic gradient in particular offers a significant reduction in complexity with an acceptable level of performance degradation.

### 6.2.6 Application

Approximating \( R_i \) may permit the real-time adaptation of the transmitted waveform in a manner such as that shown in Figure 6.2.6. In such a scheme, the interference signal \( \hat{i} \) is obtained through a separated receive channel dedicated to sampling the interference environment. Ideally, this auxiliary, receive-only channel will have thermal noise properties identical to the main receiver chain. If the adaptation algorithm is initialized with an impulse (i.e., \( w \) in Equation (6.2.6) is a vector containing all ones), then the DFT of the adapted signal \( q \) can be seen as a set of spectral weights that can be applied to any signal \( s_d \). In this case, \( s_d \) can be thought of as a template signal with a known matched filter

94
response. The spectral weights can be applied by FIR filtering $s_d$ with $q$. As a result, the convolution matrix

$$
Q^H = 
\begin{bmatrix}
q_{N-1}^* & 0 & 0 & \cdots & \cdots & 0 \\
q_{N-2}^* & q_{N-1}^* & 0 & \cdots & \cdots & 0 \\
\vdots & q_{N-2}^* & q_{N-1}^* & \cdots & \cdots & \vdots \\
q_1^* & \vdots & q_{N-2}^* & \cdots & \cdots & 0 \\
q_0^* & q_1^* & \vdots & \cdots & q_{N-1}^* & q_{N-2}^* \\
0 & q_0^* & q_1^* & \cdots & q_{N-2}^* & \vdots \\
\vdots & 0 & q_0^* & \cdots & \cdots & \vdots \\
0 & \vdots & 0 & \cdots & q_1^* & q_0^* \\
0 & 0 & \vdots & \cdots & q_0^* & 0 \\
0 & 0 & \cdots & \cdots & q_0^* & 0
\end{bmatrix}
$$

is updated at each time step, as is the matched filter $s^H$. 

95
6.3 Simulation

6.3.1 Implementation

Three variations of the update equation in (6.6) were tested in simulation, with the method of approximating $R_i$ being the only difference between the three. The first variation assumes *a priori* knowledge of the interference covariance matrix and uses the $R_i$ matrix directly. The second variation assumes no *a priori* knowledge of $R_i$, and uses the instantaneous approximation of $R_i$ given in (6.7). Similarly, the third implementation uses the recursive approximation of $R_i$ given in (6.8). The results from each of these three variations are labeled GD, ISG, and RSG respectively. It should be noted that for comparison, the simulation random number generator was initialized to the same state for each case.

6.3.2 Adaptation of a Short Pulse

Defining $\omega$ in (6.1) to be constant over the band of interest results in a short pulse transmit signal $q$. Adaptation of $q$ was simulated using the GD, ISG, and RSG implementations. Simulation results demonstrate that without regularization each implementation does indeed place the power of the transmit signal in the portion of the spectrum least occupied by the interference. (Figure 6.1) As a result the correlation between the interference and transmit waveforms can be almost entirely reduced (Figure 6.2) at the cost of increases in the matched filter peak sidelobe ratio (PSLR) and mainlobe width (Figure 6.3). It should be noted that improvement in SINR depends heavily on the extent to which the interference occupies the band of interest.[23] The results also demonstrate that the ISG implementation, which has the low computational complexity, is capable of achieving the same steady-state performance as the GD and RSG implementations.

Figures 6.4-6.7 show simulation results for the GD and ISG implementations with $p = 1$ regularization. The results show that PSLR and SINR improvement have a reciprocal relationship with the regularization step size $\gamma$. This is true for both GD and ISG cases.
Therefore, $\gamma$ must be selected depending upon whether SINR improvement or PSLR is more important for a particular application. It should also be noted that while no clear relationship is established between $\gamma$ and the matched filter response mainlobe width, the mainlobe width is evidently constrained.

Figure 6.1: Power spectral density of adapted short pulse for each implementation with no regularization.

Figure 6.2: SINR during adaptation of a short pulse for each implementation with no regularization. See Figure 6.3 for corresponding matched filter responses.

6.3.3 Adaptation of Other Waveforms

The spectral weights shown in Figure 6.1 can be applied to any waveform by FIR filtering the waveform with $q$ as shown in Figure 6.2.6. Such a process was simulated for a linearly
Figure 6.3: Matched filter response after adaptation of a short pulse for each implementation with no regularization. See Figure 6.2 for corresponding SINR.

Figure 6.4: SINR of adapted short pulse for GD implementation with various $p = 1$ regularization weights. See Figure 6.5 for corresponding matched filter responses.

Figure 6.5: Matched filter response of adapted short pulse for GD implementation with various $p = 1$ regularization weights. See Figure 6.4 for corresponding SINR.
Figure 6.6: SINR of adapted short pulse for ISG implementation with various $p = 1$ regularization weights. See Figure 6.7 for corresponding matched filter responses.

Figure 6.7: Matched filter response of adapted short pulse for ISG implementation with various $p = 1$ regularization weights. See Figure 6.6 for corresponding SINR.
frequency modulated (LFM) waveform. Assuming the template waveform $s_d$ is normalized to unit power, the SINR performance is equivalent to that of the short pulse case. Figure 6.8 shows that, in the absence of regularization, SINR improvement is achieved at the cost of matched filter response quality. Figures 6.9-6.12 show simulation results for GD and ISG implementations with $p = 1$ regularization. With regularization, the matched filter response shape better approximates that of the unadapted waveform as the regularization step size ($\gamma$) increases (Figures 6.10 and 6.12). As expected, this constraint of the matched filter response is achieved at the cost of SINR (Figures 6.9 and 6.11). In this particular case, SINR, PSLR and mainlobe width have a reciprocal relationship to $\gamma$.

![Figure 6.8: Matched filter response of adapted LFM waveform for each implementation with no regularization. See Figure 6.2 for corresponding SINR.](image)

### 6.4 Conclusion

An adaptive algorithm for waveform diversity was presented that is capable of improving the SINR performance of a radar operating in a colored interference environment while simultaneously constraining the shape of the matched filter response. The algorithm has low computational complexity, and requires no external estimation of the interference spectrum. Two independent weighting terms can be used to trade between SINR performance and matched filter response shape. Future work will consider optimal values of $\gamma$ and $p$, as
Figure 6.9: SINR of adapted LFM waveform for GD implementation with various $p = 1$ regularization weights. See Figure 6.10 for corresponding matched filter responses.

Figure 6.10: Matched filter response of adapted LFM waveform for GD implementation with various $p = 1$ regularization weights. See Figure 6.9 for corresponding SINR.

Figure 6.11: SINR of adapted LFM waveform for ISG implementation with various $p = 1$ regularization weights. See Figure 6.12 for corresponding matched filter responses.
Figure 6.12: Matched filter response of adapted LFM waveform for ISG implementation with various $p = 1$ regularization weights. See Figure 6.11 for corresponding SINR.

well as additional regularization terms to constrain waveform properties such as constant modulus.


