Low Cost Radar and Sonar using Open Source Hardware and Software.

Lance Patrick Williams

A dissertation submitted to the Department of Electrical Engineering,
University of Cape Town, in fulfilment of the requirements
for the degree of Master of Science in Engineering.

Cape Town, August 2008

Declaration

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own.

Signature of Author . . . .

Lance Patrick Williams

Cape Town

31 August 2008

Abstract

The full range of radar types and innovations can be complex and difficult to prototype, especially for institutions that wish to perform a wide range of experiments for low financial cost. Radar and sonar system development can benefit from digital technology that is powerful for research purposes, easy to use, and inexpensive. The purpose of this thesis was the development of a sonar application using the Universal Software Radio Peripheral and the GNU Radio software framework. These are Open Source tools created for the software-defined radio community. These tools provide a powerful yet flexible means to experiment with a wide range of radio frequency applications, using a minimal amount of relatively cheap hardware. In this thesis, these tools were modified from their original telecommunications purpose, to produce a sonar system that could eventually be scaled to a prototype radar system using the same device and software framework.

Acknowledgements

I would like to thank Prof. M.R. Inggs for his supervision, guidance and patience during development of this dissertation. I would also like to thank the South African National Defense Force for their financial support which made the development of this dissertation possible. I would like to acknowledge the invaluable assistance of the members of the GNU Radio development community, in particular Eric Blossom, Matt Ettus, and Jonathan Corgan. I must also thank all the members of the Radar Remote Sensing Group for being available and eager to help, in particular Regine Lord, Marc Brooker, and Stephan Sandenbergh. Finally, I would like to thank my family, whose constant encouragement and support helped me to complete this dissertation.

Contents

List of Figures

List of Tables

List of Symbols

B	—	Transmitted RF bandwidth
c	—	Speed of Sound
Δf	—	Sweeping Frequency Range
t_{pri}	—	Pulse repetition interval
τ	—	Pulse duration

Nomenclature

radar — radio detection and ranging.

sonar — sound navigation and ranging.

pulse — a finite burst of radio or acoustic wave energy.

Doppler frequency — A shift in the radio frequency of the return from a target or other object as a result of the object's radial motion relative to the radar.

Ubiquity — The telecommunications engineering goal of obtaining total network coverage in every possible location on earth.

FPGA — Field-Programmable Gate Array: a programmable array of logic gates used to create custom integrated circuits.

ADC — A/D (Analogue to Digital) converter: a device which converts an analogue signal into a digital signal.

DAC — D/A (Digital to Analogue) converter: a device which converts a digital signal into an analogue signal.

GNU — A set of freely distributed Open Source software packages.

PRF — Pulse repetition frequency.

PRI — Pulse repetition interval.

Modulator — A device or process which mixes an information signal with a carrier signal.

Demodulator — A device or process which mixes a carrier signal with a modulated signal in order to obtain the embedded information signal.

Range — The distance from a radar/sonar antenna to a target.

RCS — Radar Cross Section

SNR — Signal to Noise Ratio

Netted Radar — A radar sensor network composed of multiple nodes spatially separated in some geometric configuration.

Software Defined Radio (SDR) — An RF system design approach which implements RF tasks in software that would traditionally have been performed using analogue components.

GNU Radio — Software defined radio programming framework.

USRP — Universal Software Radio Peripheral: a computer peripheral which implements radio transceiver capabilities.

HDL — Hardware description language.

Verilog — A particular hardware description language.

Matlab — A script programming language for mathematical processing.

Python — A script programming language.

GUI — Graphical User Interface

IC — Integrated circuit: a complex circuit implemented within a single component package.

WxPython — Python extension libraries for creating graphical user interfaces to Python programs.

USB — Universal Serial Bus: a communication protocol used between electronic devices.

CIC — Cascaded Integrator Comb: a type of digital filter.

CORDIC — An algorithm for calculating sinusoidal waveforms via vector rotation.

Decimation — The process of reducing sample rates.

Interpolation — The process of increasing sample rates.

Quartus — An HDL programmer and compiler software package produced by Altera.

Daughterboard — A capability expanding add-on device for the USRP.

PGA — Programmable Gain Amplifier: An amplifier with a gain value that can be altered via programmable registers.

opamp — operational amplifier.

Chapter 1

Introduction

The rapid evolution of the telecommunications industry has lead to a great surge in the advancement of digital signal processing hardware. The demand for ubiquity in modern digital telecommunications requires hardware with ever expanding capability and flexibility to be able to implement the modern wireless network.[1]

The mass production of the devices needed for wireless communications has resulted in the miniaturisation of transceiver hardware, sub-components of which can even be implemented within a single integrated circuit. This advancement in the manufacturing processes of transceiver hardware is driven by the demand for more powerful wireless devices which can transfer data at high speeds and operate within an increasingly occupied radio frequency spectrum. The results are cheap, highly capable, highly functional, low power transceiver components that are small in size.

The implementation of transceiver hardware for modern digital telecommunications systems particularly favours digital processing approaches.[2] This is possible because of the advancement of certain key technologies:

- Processing – CPU, FPGA and DSP processing has become cheaper in terms of computational time, physical size and the cost of memory
- A/D and D/A conversion – analogue-to-digital and digital-to-analogue converters have become much faster with greater dynamic range.

The two factors listed above have also lead to greater focus on another approach to radio transceiver and wireless communication development namely, Software-defined Radio (SDR). The aim of a SDR approach to communications systems design is to implement as many of the system functions as possible in software and reconfigurable hardware. Reconfigurable hardware refers to hardware that can be redesigned and modified using a hardware description language to alter the hardware's functions and operational abilities. The advancement of processing and converter technology has allowed this design approach to be functionally applied closer and closer to the antenna in radio communication systems. Tasks such as signal definition, filtering, modulation and demodulation can all be performed in general computing software. The advantages of SDR include:

- Reconfigurability (even in real time)
- Flexibility (operational capability can be completely altered without changing hardware).

Cheap mass-produced DSP and radio transceiver component devices (some able to operate at 5 GHz), along with SDR enabling hardware and design approaches, are constantly pushing the boundaries of what is possible in the telecommunications industry. These advances and potential applications, while created for the telecommunications industry, have not gone unnoticed by the radar engineering community.

1.1 Software-defined Radar

While radars constructed with components designed for telecommunications may not provide the best performance (particularly in terms of bandwidth), such radars may still prove invaluable for the emulation and prototyping of new radar designs and principles. In addition to making use of telecommunications components to reduce cost, design time and circuit complexity, an SDR approach to radar design also offers its own means of reducing costs, and enabling faster prototyping due to its rapidly reconfigurable nature.[1]

Radar types which may particularly benefit from the ability to prototype and emulate, as described above, include the concepts of netted radar and passive coherent location (PCL) radar. Radars such as these are the topics of ongoing research. The ability to construct prototypes and emulators is needed to assist in the development of such radar/sonar systems, and to verify their theoretical and simulated performance.

These prototypes should be reconfigurable and relatively low in cost, in order to keep pace more easily with new developments. Ideally, they should be able to perform either as small scale emulators, or as more powerful and complete working systems. This should be achievable with minimal change to the hardware and software being employed.

Relatively easy development of the hardware and software should also be possible to allow new developers to quickly become familiarised with the system and its possible uses. This would allow more focus on developing the high-level concepts and architectures of emerging radar technology. A prototype/emulator like this would also make it possible for practical teaching of new developers and students of these new concepts and other existing radar systems.

1.2 The USRP and GNU Radio

This thesis focuses on the adaption of a SDR development platform called GNU Radio, and a transceiver device called the USRP, from their original telecommunication function to performing radar functions. The adapted versions were tested by implementing a small scale air sonar prototype described in later chapters.

The Universal Software Radio Peripheral (USRP) is a USB 2.0 connected computer peripheral that provides a relatively cheap hardware solution for software radio applications. It is an Open Source hardware project developed for use with GNU Radio. GNU Radio is an Open Source SDR project and software framework, which users can use to construct SDR applications. Open Source refers to hardware and software for which all the designs are freely available, and can be modified, added to, and enhanced by the general public without the need to purchase a license.

The USRP is primarily designed to implement telecommunications applications built in GNU Radio. It takes advantage of reconfigurable and flexible digital technologies such as the AD9860 DSP chips and the Altera Cyclone FPGA. These components perform high speed sampling and digital down-conversion to allow the capture of high frequency RF signals.

The USRP is priced to be affordable for amateur radio enthusiasts. Separately available interchangeable daughterboards which plug into the USRP, contain limited RF hardware. These enable the USRP to access various frequency bands which would otherwise be beyond the capability of its A/D converters to sample. This is a more cost effective solution for developing a wide range of RF applications which would otherwise require specially designed custom systems.

A USRP based system requires: a personal computer with the appropriate open source software a GNU Radio package installation; a USRP with desired daughterboards; and the necessary RF front-end devices such as amplifiers and antennas.

1.3 Objectives

The objectives of this thesis are as follows:

- Analyse the USRP and GNU Radio framework functionality. Determine their suitability for creating radar or sonar applications. Determine the limitations to creating said applications. Take note of functionality that would be useful for the creation of radar prototypes.
- Make the necessary modifications to both GNU Radio and the USRP to facilitate the creation of radar and sonar applications.
- Design and build a monostatic air sonar prototype using the USRP and the GNU Radio software. Use the prototype to observe and record echoes from various targets. This will serve as a test case to show how the USRP handles some of the basic tasks that a radar system must perform.
- Perform basic and well-known radar/sonar signal processing on the captured data sets. Use this to demonstrate the prototype's performance and behaviour.
- Draw conclusions on the prototype's performance. Make recommendations on how the prototype might be enhanced and/or modified to emulate other types of radars or sonars.
- The findings of the prototype development process are intended to be utilised for research into other radar and sonar applications using the USRP, specifically those relating to netted radar and PCL research.

1.4 Thesis Outline

Chapter 1 gives an introduction to this thesis and its purpose and what the objectives are.

Chapter 2 expands on the discussion of telecommunications components that can be used in radars. The topic of software-defined radio and the technologies that enable it are included. How SDR might service radar is discussed, GNU Radio and the USRP are introduced as being implementations of the discussed concepts. The functional design requirements of a radar application based on GNU Radio and USRP are listed. The desired performance and capabilities of such a system are proposed in a comparison to an existing netted radar prototype.

Chapter 3 provides more detailed information about the GNU Radio software framework. This will provide insight into how the sonar prototype outlined later in this dissertation was defined and implemented. The chapter then goes on to take a closer look at the USRP. It is useful to understand how the USRP achieves application flexibility by looking at the specific hardware components on the USRP and how the FPGA is configured. The USRP's capabilities and operation will be described. Features useful to radar will be highlighted. This chapter will show the limitations of the default USRP configuration and which functions would need to be altered to make radar and sonar applications possible.

The USRP is designed to work with the GNU Radio framework. How GNU Radio works and interacts with the USRP will also be covered. This will provide insight into how using the USRP and GNU Radio will affect a radar application's performance and what the system restrictions would be.

Chapter 4 will cover the details of the monostatic air sonar that was developed. The supporting GNU Radio program created for this application, as well as the FPGA level modifications that needed to be performed, will also be explained. This chapter will also describe the supporting front-end hardware that was designed and built for the sonar, including the various implementations that were attempted. The capabilities and limitations of the system produced will also be presented.

Chapter 6 will look at the various sonar tests that were performed and their intended purpose. The results will be presented and explained.

Chapter 7 summarises the work done and what was achieved. This chapter will also discuss the test results, and make conclusions about the USRP and its performance as part of a monostatic sonar system. The USRP's suitability for prototyping various other radar types will be revisited. Recommendations are made on how radar and sonar using the USRP can be further researched, as well as how the produced sonar can be improved and functionally expanded/enhanced.

Chapter 2

From Telecommunications to Radar

There are various technical advancements that have been made in the telecommunication industry, particularly in terms of capability and manufacturing. These advancements have been necessitated by the drive towards ubiquity and the consumer demand for more functionality and greater media-sharing capability. Some of these technologies have the potential to be applied to radar applications. This chapter will look at some of these technologies and what they would mean if adapted to radar. This chapter also identifies some of the aspects of radar that would benefit from a software-defined radio implementation which makes use of reconfigurable digital hardware. It also looks at the particular benefits a SDR implementation might serve for the development of radar emulators and prototypes.

2.1 Transceiver Hardware

Components available for telecommunications devices operate within the frequency ranges of certain radars. Their ability to mix signals from these bandwidths to intermediate frequencies to be digitised by high speed A/D converters optimised for mid-range performance and cost, makes these devices interesting to consider for low cost radar systems.

A powerful technology that is currently found in wireless network devices is called radio-on-a-chip (R-o-C). These are circuit components which contain many of the components that a radio transceiver requires, such as amplification and mixing, all within a single integrated circuit package. They are optimised to use low power and occupy very little physical space in order to service the mobility required in devices such as WLAN adaptors and wireless broadband modems. While not having sufficient transmission power to serve in a radar transmitter without additional amplification, they can form an important intermediary step in the development of cost effective radar transceiver hardware.

Devices such as the AR5112 created by Atheros Communications, offer R-o-C systems that are designed to work with wireless networking standards. These wireless standards occupy bandwidths at around 2.4 GHz (2.3 – 2.5 GHz) and 5 GHz (4.9 – 5.85 GHz). These are within the operational frequency ranges of S-band and C-band radars.[3]

A/D and D/A converters are of paramount importance in the advancement of digital transceivers. They are a determining factor in the performance of a transceiver as they dictate the amount of usable bandwidth which can be obtained. Interesting telecommunications products exist such as the AD9860 from Analog Devices, which combines the A/D and D/A converters with amplification, filtering, interpolation and signal mixing all within one surface-mounted integrated circuit. This device was created for use in broadband modems and is capable of theoretically converting 32 MHz of signal bandwidth in its receiver chain and 64 MHz in its transmitter chain. Also present in this device is programmable gain, making it possible to employ range-dependent gain [3] used in radar receivers to protect the dynamic range of its A/D converter.

The use of such devices does come with limitations. Since they are designed to operate in frequency bands used by established radio communication standards, the relevant portions of the RF spectrum contain much interference from other devices. They are also optimised for cost, size and power consumption, acquiring only the performance levels required for the telecommunications applications they serve.

Although bandwidths such as 32 MHz are also nowhere near the capability of wideband radar, devices with such operating capabilities might be well suited to narrowband radar applications only requiring less than 50 MHz bandwidth. The effect of narrower bandwidth negatively impacting radar performance metrics such as range resolution and target classification, can potentially be reduced by employing new radar schemes such as netted radar.

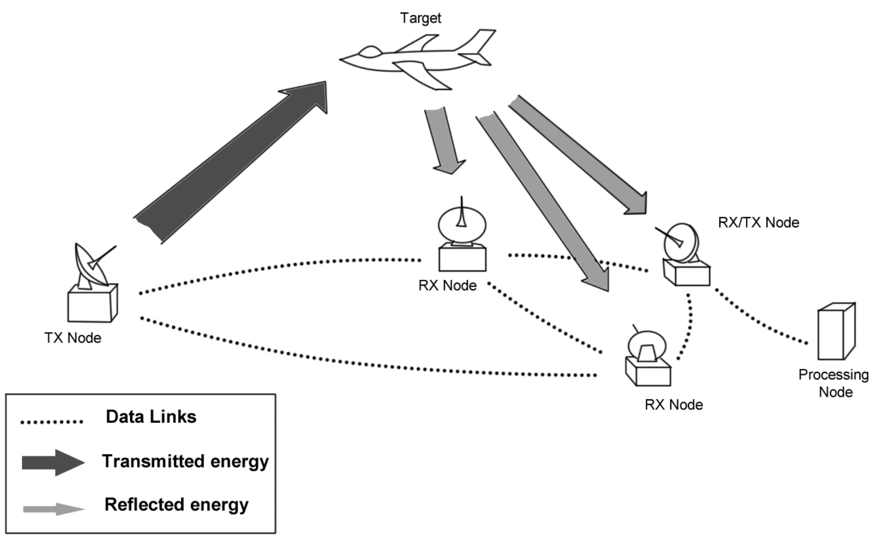


Figure 2.1: Diagram of a netted radar example containing one transmitter node, two receiver nodes, and one transceiver node linked to a processing node.

Netted radar is simply an extension of the sensor network concept. It implies a radar system composed of multiple transmitters and receivers called nodes. The transmitters and receivers may or may not be co-located. These nodes work together to more effectively observe the target area. The advantages of netted radar include:

- Receiver-only nodes are invisible to anti-radiation weapons and are less prone to targeting by electronic countermeasures such as jamming.
- Certain spatial configurations can ease design constraints or favourably alter parameters on particular nodes.
- Receiver nodes can be placed far from the transmitters and other sources of radio interference. This would provide protection from the transmitter and allow for more flexible node system parameters.
- Multiple nodes introduce redundancy, which improves the total system's survivability and reliability.
- Multiple nodes are able to collect more total scattered radio energy, which can potentially increase the system's overall sensitivity.[4]
- There is a potential increase in the probability of detection of targets, since environmental effects and interference, which would degrade the RCS of the target, may not affect all receivers within the system at once. This is also true for a fluctuating target RCS, since it is unlikely that all nodes will experience the same nulls in the RCS simultaneously.[3]

- Multiple nodes in suitable spatial configurations can also counter stealth technologies, since attempts to disperse, mask or otherwise alter the RCS of the target, may not affect all nodes in the system simultaneously.

- Multiple nodes separated over suitable distances give the system more perspectives of a target. This provides more information on the target shape, which can improve target classification and potentially reduce false alarms.[4]

An analysis of the radar equation adapted for netted radar [4] shows that under ideal conditions, the SNR can improve by as much as the square of the number of nodes in the system, if coherency can be preserved in collective receiver data processing. Thus, as more radar nodes can be added to the network, system performance can begin to surpass that of a non-networked radar system. This represents a possible means of regaining useful radar performance while still using telecommunications components, by increasing the number of nodes while preserving coherency.

2.2 Software-defined Radio

2.2.1 Concept

Software-defined radio is a term which describes the use of software and reconfigurable digital hardware to perform radio transceiver component tasks previously performed by analogue components and subsystems.

Consider a simplistic overview of a radio system. A signal to be transmitted would be modulated, amplified and applied to an antenna. The radio wave would propagate through the air, and a signal would be received at the receiving antenna. This received signal would be subjected to amplification, filtering, and demodulation.

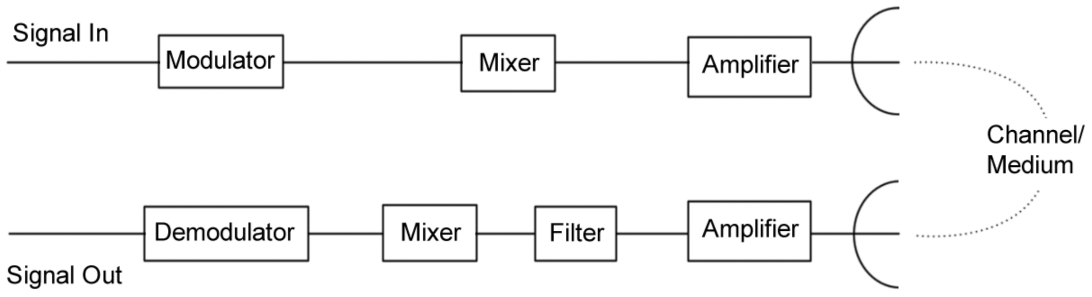


Figure 2.2: Basic diagram of a radio transmission and reception system.

Thus, within this radio system, there are some easily identifiable tasks which must be performed such as modulation/demodulation, filtering, amplification and mixing. [5] There may also be many stages to each of these tasks, but these too can be clearly identified and described. A software-defined radio system would implement as many of these tasks (or stages in these tasks) as possible using general computing software processing. SDR has been employed in systems such as the Joint Tactical Radio System (JRTS), which takes advantage of the adaptability and compatibility of SDR.[6]

2.2.2 Hardware for SDR

Components such as mixers, amplifiers and filters are also found in radar systems. Therefore, an SDR approach may also be applied to radar designs. Due to the emergence and continual evolution of high speed digital hardware, software-based solutions to radar problems are becoming more viable.

The goal of SDR receiver design is to digitise received signals as close to the antenna as possible. The limitations of ADC hardware must be balanced against the desired operational bandwidth of the receiver and signal fidelity. Similarly, DAC hardware is considered versus transmission bandwidth and output signal quality. Radars typically operate in frequency bands beyond the compatibilities of cheaper ADC technology to sample the received signals directly. This implies some degree of front-end hardware is required to mix received signals to intermediate frequencies that the ADC can sample effectively.[6][2] Section 2.1 of this chapter looked at how hardware components designed for telecommunication may fulfil this requirement.

Once the signals are digitised, the component most commonly used to transfer the samples to a computer system to be processed using software is the Field Programmable Gate Array (FPGA). This device makes an excellent development tool. Manufacturers include a wide variety of additional features and functionality besides the FPGA's reconfigurable logic structures. These include features such as A/D and D/A conversion and integrated circuit elements which can perform data transfers via well-known protocols such as I2C and Ethernet.

Their reprogrammable functionality and speed make FPGA's ideal for SDR applications. They are also useful to designers in that pre-designed cores, which perform various functions such as FFT or CORDIC, can be acquired for free (in some cases) and incorporated into other designs. This can greatly reduce the time needed for development of a new application.

While perhaps not being truly pure components of the ideal software radio, in that they are still hardware components and cannot simply be treated as another stage of software development using the same design method, they do provide many of the advantages of software which software-defined radios aim to exploit.[7] By integrating FPGA's with existing transceiver hardware, a reconfigurable element is added, which combines the speed and parallelism of hardware with the flexibility of software.

2.3 SDR and Radar

The main advantage of SDR for a radar system is its reconfigurability, both on the fly and during design and redesign. A SDR application can completely change its capabilities and even its operational ability, simply by altering the defining software code. Implementation of dynamic radar functions such as staggered PRF, frequency agility and complex modulation and code modulation schemes within a varying array of radar types, can be approached as software problems.

The capabilities of hypothetical SDR based radar might include the ability to completely change operating modes from surveillance, to tracking to passive operation. Multiple tracking algorithms and filters could be tested and altered in real time. Treating such tasks comparatively, as interchangeable blocks, the radar can potentially be subjected to an evolutionary program, which can analyse, test, discard and interchange various function types and combinations autonomously to find the most effective system.

The above is simply an example of the type of systems that could be implemented with the ability to define a radar application in software and reconfigure its operation simply be altering or replacing the software. The example is rather ambitious in terms of what is possible, even with today's processing methods and hardware configurations.

In a situation where the ability to change rapidly between operational functions and features is not required and processing power is compromised, SDR becomes valuable for prototyping, emulation, and educational demonstration. In this case the same hardware can be used to implement multiple radar schemes, or to adapt quickly to new data, or test new theories. This can greatly reduce the costs in terms of designing or purchasing new hardware and in application implementation where software design is well understood.

Designing the logical system and actually implementing it, can potentially be complicated unless the design process can be abstracted. In the case of abstraction however, certain considerations of the underlying systems and technologies which are abstracted may be difficult to resolve.[6]

For the purpose of quickly gaining the knowledge and ability required for developing applications for teaching, testing and emulation, it would be useful if an easy-to-use framework for developing applications already existed. This thesis looks at using an existing Open Source SDR application building framework called GNU Radio to implement radar applications. This thesis also examines an Open Source hardware transceiver device called the USRP, to see if it can be modified to implement pulsed radar applications. The USRP is a computer peripheral device designed to implement applications developed with GNU Radio, and it employs many of the telecommunications hardware components discussed in previous sections.

2.4 Radar using the USRP and GNU Radio

The USRP and GNU Radio are tools created primarily for telecommunications orientated SDR applications. The main work of this thesis is a study of the theoretical capabilities of the two systems, modification of the system functionality and then implementation of a basic test case. For comparison, this project was envisioned to provide a hardware and software alternative to the radar nodes used in a prototype netted radar developed at The Electrical Engineering Department at University College London (UCL).[8, 9] Their system is comprised of three transceiver radar nodes which are connected via twisted pair cables. A 200 MHz distributed clock synchronises the nodes, which operate at 2.4 GHz with 50 MHz bandwidth. A low frequency clock pulse is also distributed to the nodes and is synchronised with the radar system's PRF, which is variable from 1-10 kHz. The system can transmit various pulse waveforms with adjustable pulse durations of 10 ns to 20 ns.

Thus, in analysing the features of the USRP and GNU Radio, they will be compared to the capabilities of the netted radar nodes described above. This thesis expands on the paper submitted to the IET Radar 2007 conference.[10] This project aims to eventually make a contribution to the development of netted radar theory through facilitating the creation of netted radar prototypes. A USRP and GNU Radio implemented netted radar application is envisaged to be comprised of multiple USRP transmitters and receivers connected to a central data processing point. The system would take advantage of several features of the USRP which will become evident in Chapter 3.

To have modified the USRP and GNU Radio successfully for radar applications, the following functions and abilities were defined as being necessary:

- Definition of a classical radar waveform, and flexibility in waveforms that can be implemented.
- Accurately produce a radar waveform.
- Provide precise PRF definition and control.

- Accurately capture a received waveform with control over receive windows and the starting time of receiver digitisation.
- Provide precise timing control relative to PRF in order to resolve phase and range information.
- Provide means of reliably archiving and displaying received data.
- Provide means of accurately processing received data.

In order to test the basic functional elements of the modifications made, a small-scale monostatic sonar application was implemented using the USRP and GNU Radio. This sonar would employ small, cheap ultrasonic transducers as its transceiver antennas, and very minimal amplification and filtering circuitry. By minimising the amount of circuitry required, more time could be spent focusing on the functional changes that needed to be implemented within the USRP and GNU Radio. The ultrasonic transducers that were used are also fairly robust. A sonar with a low operating frequency would also be less sensitive to front-end circuitry design considerations such as noise, attenuation and phase shifting.

Chapter 3

GNU Radio and USRP

This chapter provides more detailed information about the GNU Radio software framework. This will provide insight into how the sonar prototype outlined later in this dissertation was defined and implemented. The chapter then goes on to take a closer look at the USRP. It is useful to understand how the USRP achieves flexibility in its applications by looking at the specific hardware components on the USRP and how the FPGA is configured. The USRP's functionality and operation will be described. This chapter will also show the limitations of the default USRP configuration and what functionality would need to be altered to make radar and sonar applications possible.

The USRP is designed to work with the GNU Radio framework. How GNU Radio works and interacts with the USRP will also be covered. This will provide insight into how using the USRP and GNU Radio to emulate a radar application would affect the performance and what the system restrictions would be.

3.1 GNU Radio

3.1.1 Overview

GNU Radio is an open source project which allows amateur radio enthusiasts to gain experience using software-defined radio. It is a software framework (developed in Linux) which forms a basis for creating radio applications and designing their functionality.[11]

To build an application with GNU Radio, a user combines various data processing elements. These processing elements are divided into signal source blocks, signal sink blocks, and signal processing blocks.

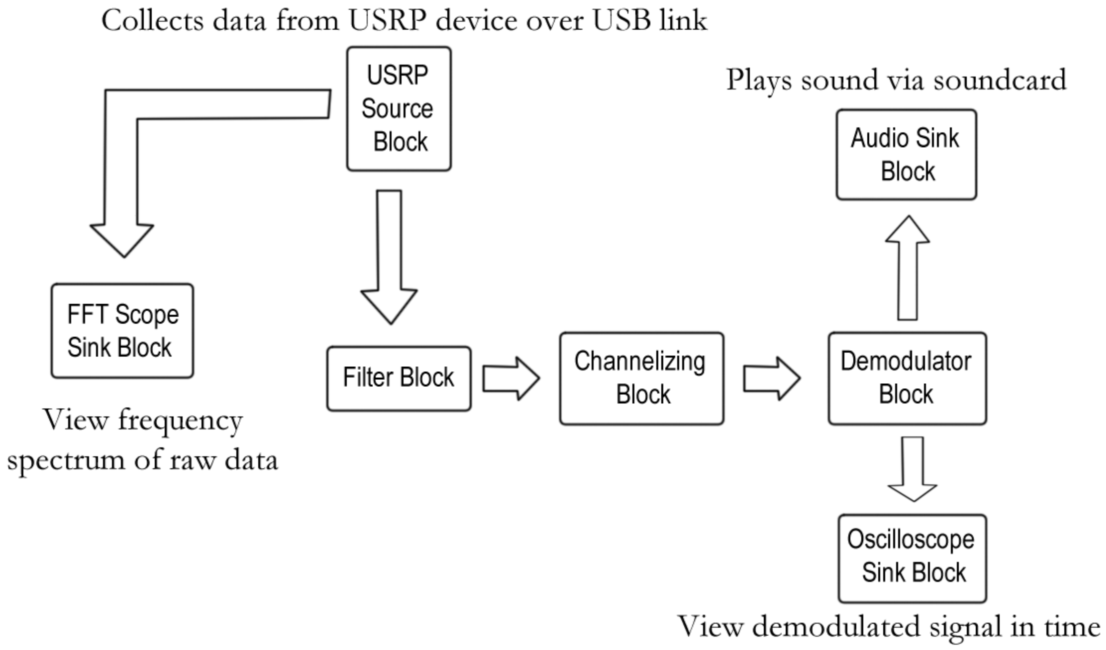

Collects data from USRP device over USB link
USRP Source Block
Plays sound via soundcard
Audio Sink Block
FFT Scope Sink Block
Filter Block
Channelizing Block
Demodulator Block
View frequency spectrum of raw data
Oscilloscope Sink Block
View demodulated signal in time

Figure 3.1: Diagram showing how a FM radio application is logically defined using the GNU Radio framework.

3.1.2 Signal Sources

Signal source blocks are the origin of user defined signals. These can be digitally defined and created signals, a data file, or signals gathered from some external source, such as a hardware element (e.g. a microphone). By abstracting any signal input as simply a signal source, users can simplify the design process.

3.1.3 Signal Sinks

Signal sinks consume signals, usually for the purpose of output. Objects abstracted to signal sinks include graphing modules, binary files, or even sound-cards. Devices such as the USRP are also abstracted as signal sinks from within the GNU Radio flow graph.

GNU Radio makes use of the Wx widget libraries for Python. Using these libraries allows users to make use of Graphical User Interfaces (GUI) to make using the resultant GNU Radio applications easier. Combined with the graphing libraries, these GUI interfaces can be used to plot data. They are also useful for educational purposes, as they provide an easy way to demonstrate RF technology and concepts.

The GUI code is treated as a signal sink within GNU Radio, with processed data streams being logically linked to various output blocks from within the graph model. Since many of these GUI contain controls as well, they can also be used to exert active control over signal processing blocks.

3.1.4 Signal Processing Blocks

Signal processing blocks work on the data stream between signal source and signal sink. Among the many existing processing blocks, GNU Radio includes:

- filters
- amplifiers
- demodulators/modulators
- FFT
- decoders/encoders.

The data processing elements described above are defined using C++. Users are able to mimic and import base classes to write unique custom processing blocks. All the framework code is viewable, so even the particular algorithms being employed can be examined.

While the processing elements are defined in C++, a user will actually build the desired application in Python. All the high level elements are defined, joined and coordinated with the Python scripting language.

3.1.5 Developing with GNU Radio

Through the GNU Radio mailing list, users can receive support for developing their own applications and understanding the details of the existing system.

GNU Radio itself can be obtained as either a compressed file, or via an online subversion repository as new versions are released and development continues. The compressed file contains GNU Radio library files and example files.

Open Source development keeps driving the development of new add-ons. Apart from existing processing blocks, users can make use of the existing files as a template for developing more specific functionality such as custom waveform generators or filters. Add-on code files are compiled and installed into the GNU Radio install tree. From there they become available to applications as signal sources, sinks, or processing blocks.

3.1.6 GNU Radio and USRP

In order to use GNU Radio for practical radio applications, a suitable hardware interface to RF transmitting and receiving is required. One of the solutions to this requirement is the Universal Software Radio Peripheral (USRP).[12] If one considered RF systems in terms of the Open Systems Interconnection Basic Reference Model (OSI) the USRP would perform the functions of the hardware and data link layers.

Using the USRP's flexible functionality, applications such as the following have been achieved by others using GNU Radio:

- FM Radio transmitter and receiver
- TV receiver
- GMSK transmitter and receiver
- GPS tracker.

This wide variety of applications was achieved using the USRP (with correct antennas and the necessary additional front-ends) and GNU Radio. Similarly the sonar application developed for this thesis was implemented using the USRP and GNU Radio.

3.2 The Universal Software Radio Peripheral

3.2.1 Overview

This section will focus on the Universal Software Radio Peripheral (USRP), taking a simplified look at how it works. Its purpose is to provide a physical layer solution to a wide range of primarily telecommunications applications that require the ability to transmit or receive radio frequency signals.

The GNU Radio framework replicates both the application and a data layers in terms of the OSI Model. The concept this conveys is the design independence of each layer. The USRP and GNU Radio are, in fact, designed to be able to operate independently of each other. It is primarily the requirements of their interfaces that must be satisfied. This allows designers to freely design specific functionality within only GNU Radio, or only within the USRP, taking care simply to make sure that the interface conventions are maintained.

On its own, the USRP is simply another USB device. It is possible to design and implement interfaces to it via programs or frameworks other than GNU Radio. This can be more complicated however, as the primary focus of the GNU Radio and USRP developers is for these two projects to integrate with each other.

It should be noted that the USRP was designed primarily for telecommunications applications. The underlying implementation of its functionality is not suited to time critical, and time managed systems such as radar or sonar.

3.2.2 Construction

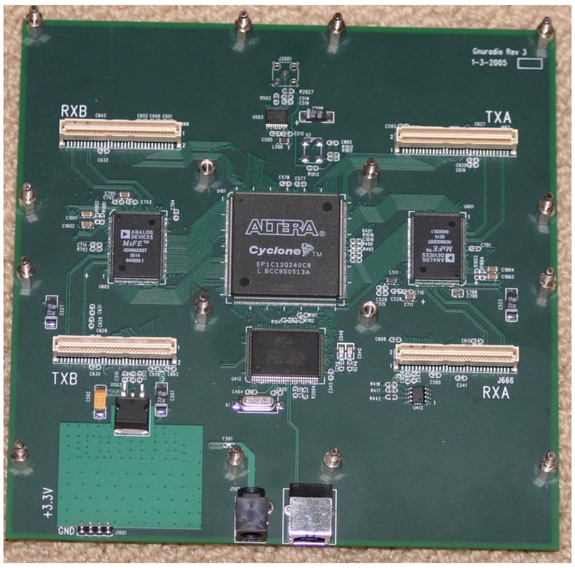


Figure 3.2: Picture of the Revision 3 Universal Software Radio Peripheral.[13]

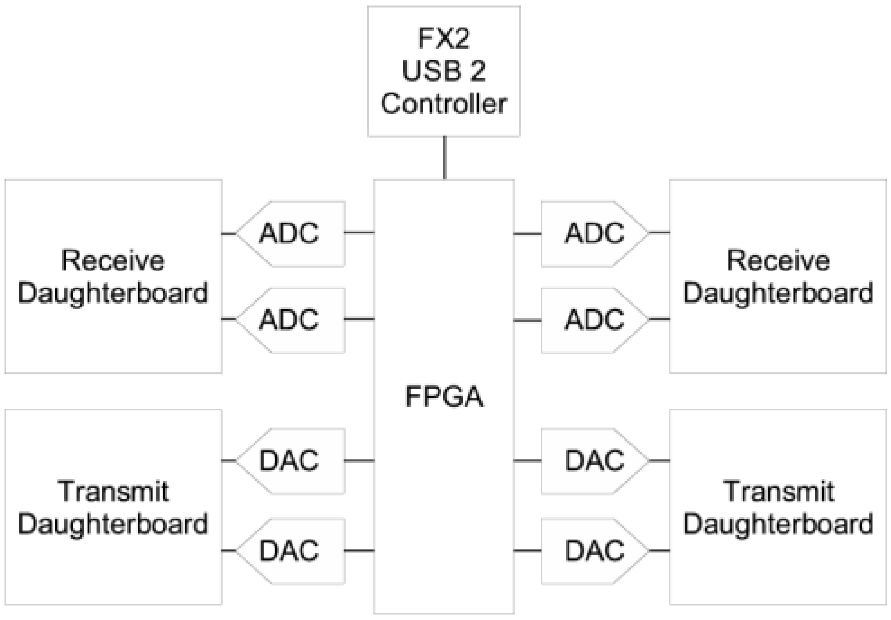


Figure 3.3: Basic diagram showing a system level representation of the USRP.[13]

The USRP connects to a host computer via USB 2.0. The USB protocol is managed by the Cypress FX2 chip. This IC provides a simple means for a hardware developer to add USB connectivity to their hardware. Upon plugging in the USB connection and powering up the USRP, the host computer will flash firmware onto the FX2 chip. This will initiate setup communications with the host computer to determine the link type, and what link settings the connected device will require. For the USRP, the USB link is set up for USB 2.0 transfer rates. The maximum transfer rate the USB link can nominally support is 32 Mb/second.

On the USRP, digital signal processing (predominantly sample rate changing) is performed within the Altera Cyclone FPGA [14]. Specifics about the FPGA's role on the USRP and its configuration are explained later in this chapter.

The signal paths are completed by 4 ADCs and DACs which are integrated on the two AD9860 chips [15] physically located beside the FPGA.

In the transmitting path, these chips contain a Hilbert filter, an interpolator (maximum factor of 4), and a digital quadrature mixer. The AD9860 performs 14 bit D/A conversion at 128 M samples/sec and has a programmable gain amplifier on the output.

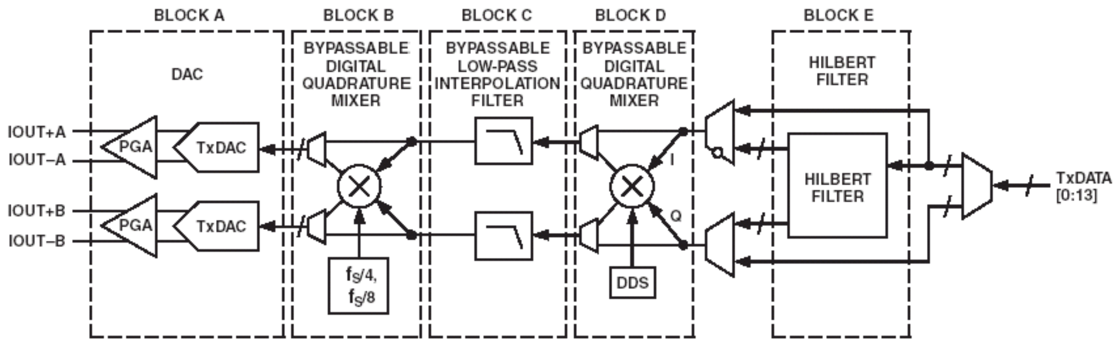


Figure 3.4: Extract from the AD9860 datasheet showing a functional depiction of the AD9860 transmit chain.[15]

In the receiving path, the AD9860s have buffers and programmable gain amplifiers on the inputs. The onboard ADC converts to 12 bit samples at 64 M samples/sec. It also contains a Hilbert filter which can provide some degree of image rejection. The Hilbert filter is deactivated in the standard USRP configuration. The AD9860 can work with both real and complex signals/sample streams and is capable of multiplexing its inputs and outputs.

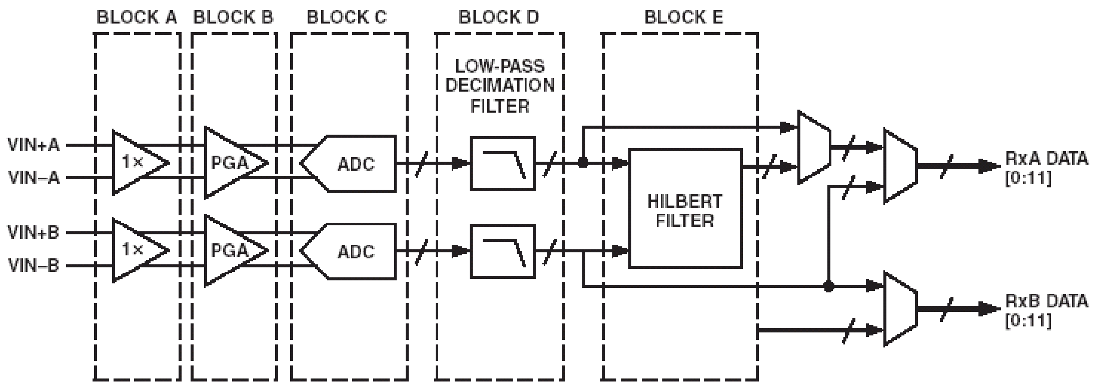


Figure 3.5: Extract from the AD9860 datasheet showing a functional depiction of the AD9860 receive chain.[15]

These chips provide the USRP with two complex transmit channels and two complex receive channels. They provide a theoretical unambiguous transmitter bandwidth of 64 MHz and a receiver bandwidth of 32 MHz according to the Nyquist theorem. Their clocks are derived from the USRP's 64 MHz crystal oscillator. The AD9860s have their own clock doubling circuitry to generate the clock for the DAC. Therefore, if the USRP's clock circuitry is changed, the AD9860 clock rates will change as well.[2]

To access the maximum possible USB 2.0 transfer rate, all data transfer must be done in packet bursts. The USB 2.0 requires that the host computer and the connecting device, package all data into 512 byte packets before burst transfer occurs. A data set is not transferred until there is at least 512 bytes of data to send.

The USRP consumes and produces 16 bit complex samples. Thus, 256 samples would constitute 1 packet. The USRP also interleaves I and Q samples into one data stream. Thus, within any single packet comprised of 256 samples, there would be 128 real samples and 128 imaginary samples.

Add-ons and Additional Features

To receive and transmit at frequencies above the AD9860 limitations, the USRP has ports for add-on daughterboards. These daughterboards contain any additional analogue circuitry needed to mix desired signals down to frequencies the ADCs can sample without aliasing, or up to frequencies beyond what the DACs can produce. Some of the daughterboards available for the USRP are described in later sections.

In newer revisions of the USRP, the designers have accommodated connection pads on the printed circuit board (PCB). Most notable is the ability to solder on a SMA connection to replace the built-in clock circuitry. This allows an external clock signal to be injected into the board, or for the board's own clock to be shared. This adds functionality which lets multiple USRP boards run off a signal clock allowing for a degree of phase coherence among multiple USRPs working together.

3.2.3 Specifications Summary

The basic setup of the USRP is as follows:

- 4 DACs, 14 bit, 128 MSamples/sec, interpolators and quadrature mixers
- 4 ADCs, 12bit, 64 MSamples/sec.
- FPGA based Cascaded Integrator Comb (CIC) filters to perform interpolation and decimation.
- Numerically Controlled Oscillator, based on an implementation of the CORDIC algorithm.
- Halfband receiver filter (decimation factor 2).

The USRP FPGA transfers 16 bit complex samples at variable rates. The USB link supports up to 32 Mb/sec transfers. Thus, using 16 bit samples, the link supports 8 MHz bandwidth. Sample sizes at the USB link can be halved to 8 bit samples, allowing the link effectively to support 16 MHz bandwidth.[16]

The FPGA is clocked at 64 MHz. The Cypress FX2 is clocked at 40 MHz. The AD9860 clock is derived from the same crystal oscillator and provides the FPGA clock (hence the 64 MSamples/sec sample rate). These chips contain their own internal clock doubling circuitry that provides a 128 MHz clock for the transmit chain. Certain daughterboards such as the RFX and TVRX daughterboards contain their own crystal oscillators.

3.3 GNU Radio Interface

Apart from the tools and methods for creating applications in Python, GNU Radio also has various software interfaces to the USRP. The various hardware parameters are adjustable via registers.

The bulk of adjustable parameters are within the FPGA. The process of changing registers is, as yet, not in sync with the data rates within the FPGA. Thus, parameters which affect the transmission or reception of samples cannot be changed in real time. Before start up, or when the application is paused, are the best times to change settings. Newer versions of FPGA builds and GNU Radio are being revised in order to support inband signalling, which would allow register updates synchronised with sample transfers. Settings which can be changed include decimation and interpolation rates, sample sizes, number of channels and channel multiplexing.

The FPGA uses serial i2c to communicate settings to connected daughterboards. These interfaces are also adjustable from GNU Radio. These allow adjustment of gain values and transmit/receive switching for daughterboards which support that functionality. The USRP can also perform certain automatic functions with particular daughterboards via these interfaces, such as automatic gain control in the TVRX daughterboard.

3.4 Transmit Chain

The digital signal processing performed by the USRP can be divided into two parts: the transmit chain and the receive chain. This section will take a closer look at the transmit chain.

3.4.1 FPGA

Beginning from the USB connection point, sample data received from the host computer is firstly de-interleaved. Depending on the daughterboard being used and the user settings, this data stream is divided into channels and then into I and Q streams. These samples are buffered in order to compensate for the differing clock domains of the FPGA and the USB controller.

The separated channel data streams are then piped to Tx chain modules. There is one Tx chain module per complex channel. Each of these modules contains a Cascaded Integrator Comb (CIC) interpolator.[17] The interpolation value set in the FPGA is extrapolated by the GNU Radio software, based on the sample rate the user requests.

The AD9860 chips are configured by default to perform interpolation by a factor of 4 on the transmit sample stream. The FPGA CIC filter is configured to interpolate by 1/4 of the user requested rate. The FPGA register that stores this resultant value is 7 bits wide, giving a maximum interpolation value in the FPGA of 128.

This combined with the interpolation rate of 4 in the AD9860, gives a maximum possible interpolation rate of 512 that the user can request from a GNU Radio application script. This also means that the user can only select an interpolation rate that is divisible by 4.

The rate is chosen such that the DACs are always provided with samples at exactly 32 Msamples/second for each channel.

From the Tx chain module, the interpolated sample streams are directed to a user controlled multiplexer. The user can select between 4 DACs (2 per AD9860 chip) to which to route the streams. Each DAC is committed to one sample stream. This means that I and Q sample streams will each be handled by a separate DAC. Thus, the USRP can manage 2 independent complex channels, or 4 real channels.

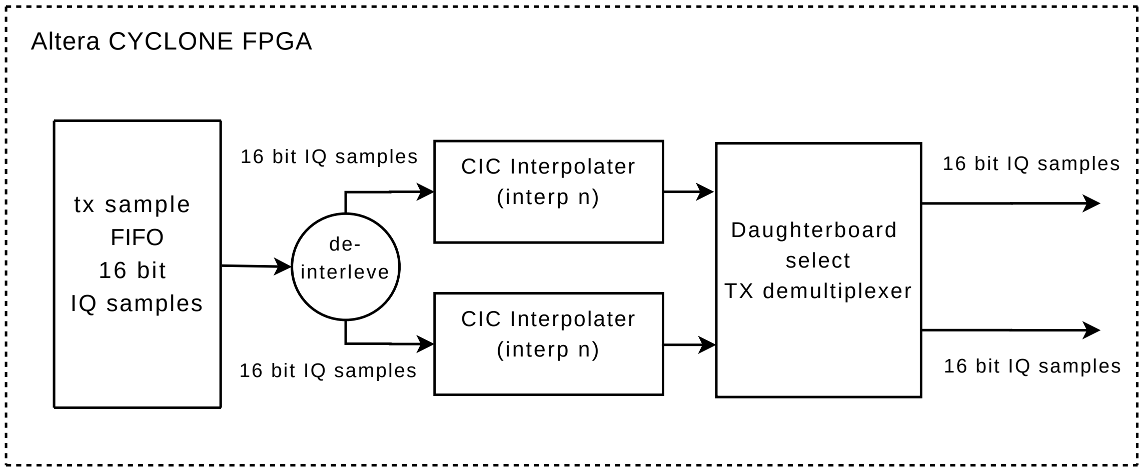

Altera CYCLONE FPGA
tx sample FIFO 16 bit IQ samples
de-interleve
16 bit IQ samples
16 bit IQ samples
CIC Interpolater (interp n)
CIC Interpolater (interp n)
Daughterboard select TX demultiplexer
16 bit IQ samples
16 bit IQ samples

Figure 3.6: Diagram showing a system level view of the transmit chain within the USRP FPGA.

3.4.2 AD9860

From the FPGA the sample stream is sent to the AD9860. These DSP chips contain a quadrature modulator, an interpolator, and a programmable gain amplifier (PGA) which provides 20 dB of gain. The DAC runs at 128 Msamples/second.

3.4.3 Daughterboards

The daughterboards are basic RF front-ends specifically designed to expand the USRP's capabilities. They include various functions depending on the frequency band to be accessed and connection points for antennas. The transmitter daughterboards currently available include:

Basic Tx: This is the basic board used for testing the USRP. It contains no RF hardware besides an output transformer which simply blocks DC signals. This board does however, have numerous debug output pins, making it very useful for debugging purposes. A user can route any internal FPGA signal to one of these 16 output pins in order to debug custom FPGA builds and modifications.

LFRX: These are daughterboards specifically made to operate at low frequencies right down to DC.

RFX: This series of daughterboards contains onboard oscillators and mixers, additional gain and custom controls such as transmit-receive switching. These boards offer access to varying frequency bands right up to 2.9 GHz provided by the RFX2400. These boards usually span transmitter and receiver slots on the USRP forming a transceiver unit. Their layouts allow for modifications such as output ports for access to the daughterboard's independent clock.

3.5 The Receive Chain

3.5.1 Daughterboards

The following receiver daughterboards are available for the USRP:

Basic Rx: Like the Basic Tx daughterboard, it contains no special RF components or functionality. It serves the same debugging purpose as the transmitter variant and has 16 debug IO pins that can be used to monitor selected signals within the FPGA, or to inject a digital signal into it.

RFX: As mentioned in the previous section, these boards span both transmit and receive slots on the USRP. Thus, they have transceiver functionality. This allows for reception of various frequency bands depending on which model daughterboard is used. Among their features is onboard I-Q demodulation, IF down-conversion and gain control.

DBSRX: This receiver board is made specifically to receive signals between 2.2 and 2.4 GHz. As with the RFX range, it has I-Q demodulation, IF down-conversion and gain control.

TVRX: This is an older model daughterboard which is still supported. It receives signals between 50 and 800 MHz. It supports up to 6 MHz bandwidth. It features IF down-conversion, I-Q demodulation and automatic gain control. It is ideal for receiving FM radio stations and has enough bandwidth to capture a TV channel image signal along with its audio component.

3.5.2 AD9860

The USRP receive chain begins at the AD9860 once more. On the receive side these chips contain a PGA which can provide 20 dB gain and a Hilbert filter, which is disabled in the standard configuration as it lowers the maximum sample rate. The ADC runs at 64 Msamples/sec and delivers 12 bit samples to the FPGA. There is one ADC dedicated to each channel, though the user can choose whether they are connected or not, via the multiplexer within the FPGA. The channels in this case are representative of either 4 real channels, or 2 complex channels.

3.5.3 FPGA

Once the interleaved samples reach the FPGA, the DC offset introduced by the ADC is removed. The samples are also adjusted to signed 16 bit sample streams. The logical module within the FPGA which performs this alteration is known as the ADC-offset module.

The sampled signal undergoes digital down-conversion (DDC) in the Rx chain module.[18] Signal mixing is done using a numerically controlled oscillator (NCO). This NCO module is implemented using a CORDIC algorithm to calculate and generate the local oscillator signals. The mixed signal is then applied to a CIC decimator to reduce the sample rate. The limiting factor is the eventual bandwidth that the USB link can support. Apart from the user specified decimation rate, the standard USRP FPGA configuration decimates by at least 2 by default, though this can be deactivated if the FPGA code is recompiled.

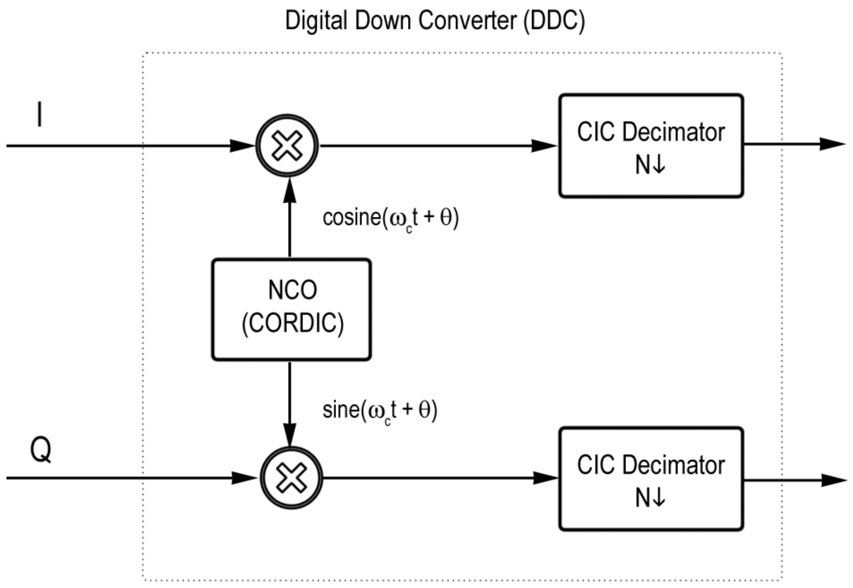


Figure 3.7: Diagram of the digital down converter implemented in the FPGA.

The reduced sample rate is then sent to the Rx buffer module. Here the samples are interleaved and packed into a buffer awaiting transfer to the host computer. In this module, it is also possible to reduce the sample size. By reducing the samples from 16 bits to 8 bits, the effective bandwidth that the USB link can support is increased. This increase comes at the cost of dynamic range.

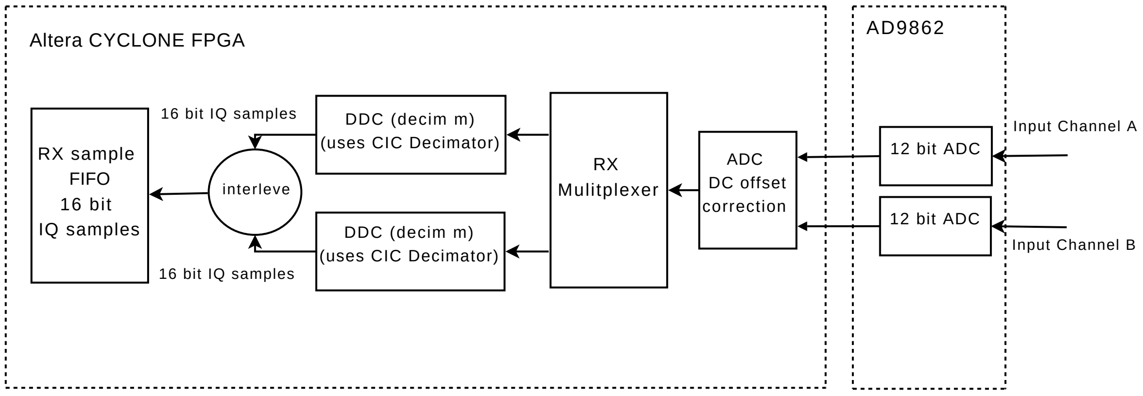


Figure 3.8: Diagram showing a system level view of the receive chain within the USRP FPGA including the AD9860 chips.

3.6 Working with the FPGA

The USRP's FPGA is programmed when a new application is launched from the host computer. Programs are not retained when the device is switched off. When connecting the USRP power supply and connecting the USB link, the Cypress chip is configured via a firmware download from the host computer. Drivers are available within Linux that recognise the Cypress chip and perform the firmware writing functions. This process initialises the USB link and configures the transfer speeds and data packet sizes.

Once this is done, the FPGA can be programmed. The FPGA is programmed with a pre-compiled binary file with the extension .rbf. When a GNU Radio application is programmed, the user can select the binary file they wish to program the FPGA with. Only when the application is executed, will the FPGA be programmed. Once it has been fully programmed, the application will begin and transceiver data will begin to flow.

The FPGA binary files used with the USRP are programmed in Altera's Quartus Software Program. The HDL language used is Verilog. All the standard code used by the USRP FPGA, in addition to the basic Quartus project files used to compile it, are bundled with the GNU Radio package.

Using Quartus, the code can be viewed and modified as a user wishes. Modified Verilog code with custom functionality can then be compiled in Quartus to produce a custom FPGA binary file for the USRP. This file is all that is needed to program the FPGA. Once copied to the correct system directory, the GNU Radio user can select this modified binary file to program the USRP FPGA. In this way, custom FPGA functionality is easily shared among GNU Radio and USRP users for use in their own applications.

3.7 Conclusion

GNU Radio includes many useful features for building SDR applications. The code libraries are easily viewable and can be added to by following the documented instructions. Access to threading control however is very difficult. Sample timing within GNU Radio is controlled by one thread managing program, which cannot be easily modified.

The USRP is a very flexible tool for implementing RF applications. It is however, primarily designed for telecommunications systems. It lacks strict management of timing samples within the FPGA. The ability to manage and record the timing of events within the FPGA is vital for implementing radar and sonar applications. Other disadvantages in the USRP include:

- Registers not written in sync with sample rates - This makes implementing dynamic changes to signal outputs in real time difficult
- Certain daughterboards lack clock outputs - in certain daughterboards such as the TVRX, access to the onboard oscillator can be difficult, which needs to be overcome if phase locking between the transmitter and receiver is to be achieved.
- FPGA utilisation is high - in the standard USRP FPGA build, when 2 complex receiver channels and 2 complex transmitter channels are implemented, adding additional processing within the FPGA becomes virtually impossible due to lack of space.

The later chapters will look at how the components of the USRP and their layout affect the development of USRP based radar and sonar applications. The potential for simple modifications and the type of applications that might be achieved will also be discussed.

Chapter 4

Development of a Monostatic Sonar Application

This chapter will cover the details of the monostatic air sonar that was developed. The supporting GNU Radio program created for this application, as well as the FPGA level modifications that needed to be performed, will also be explained. This chapter will also describe the supporting front-end hardware that was designed and built for the sonar, including the various implementations that were attempted. The capabilities and limitations of the system produced will also be presented. The output scripts and code files are available on the attached compact disk, the directory structure of which can be viewed in Appendix C.

4.1 Test System Goals and Requirements

To emulate a radar node using the USRP, a small scale sonar application had to be developed. This same system had to be adjustable to serve as a back-end to a future radar application by modifying various parameters.

The system had to make use of 40 kHz ultrasonic transducers [19]. These devices and the accompanying amplification and filtering circuitry had to be able to emit linear chirp pulses with a centre frequency of 40 kHz and a bandwidth of at least 2 kHz (determined by the transducer characteristics). The only other hardware items to be used were the USRP and a desktop computer using a Linux operating system.

The USRP was utilised with the Basic daughterboard pair. Some modifications had to be made to the USRP FPGA in order to make pulsed sonar and any eventual radar applications using this device possible. These modifications are described fully in the sections that follow later in this chapter.

A GNU Radio program had to be created utilising the standard libraries and utility blocks provided with the standard GNU Radio release. This program is described in section 4.3.

4.2 System Specifications

Table 4.1: USRP-based sonar application parameters.

Centre frequency	40 kHz (Adjustable)
Bandwidth	2 kHz (Adjustable up to 32MHz)
τ	1 ms or 500 μs
PRI	10 ms (Adjustable up to 67.108 sec).
Delay before initiating data capture.	0 sec (Adjustable up to 67.108 seconds)
Receive Window size	5 ms (Adjustable up to 67.108 seconds)
FPGA based waveform sample RAM	2 kB
No. of samples that can be stored (16 bit complex samples)	256 or 512
Selectable hardware interpolation rates	4 - 512 (Multiples of 4)
Selectable hardware decimation rates	4 - 256
Maximum USB link transfer rate	32 MB/sec
USB link bandwidth capacity (16 bit complex samples)	8 MHz
Selectable sample sizes	8bits or 16bits
Samples gathered per test	128000 (Completely adjustable)
Hardware gain range	0 - 20 dB

The expected range resolution of the resultant system was calculated as 8.5 cm using equation 4.1.

$$\text{Range Resolution} \quad = \tfrac{c}{2xB} \tag{4.1}$$

Where: c = 340 m/s (speed of sound in air)
B = pulse bandwidth

For the purpose of testing this application, a pulse duration of 1 millisecond was predominantly used to transmit more energy. The Pulse Repetition Interval (PRI) largely used for testing this application was 10 ms. The unambiguous range was calculated as 1.7 m using equation 4.2.

$$\text{Unambiguous range} = \frac{c \times \text{PRI}}{2} \tag{4.2}$$

Where: c = speed of sound in air.

The dead range for the system was calculated as 17 cm using equation 4.3.

$$\text{Dead range} = \frac{c\tau}{2} \tag{4.3}$$

Where: τ = Pulse Duration

4.3 The GNU Radio Control Program

The sonar application program was developed as a Python language script and uses only the various processing, sink and source blocks available within the standard installation of GNU Radio 3.01. The script can be viewed in Appendix A and its development is described in the following sections.

4.3.1 GNU Radio Transmitter Overview

Waveform Definition

The application creates a baseband linear chirp in software and generates a real sample set. Using GNU Radio processing blocks, a complex sample set is then generated. The waveform parameters are modified as follows:

The frequency range of the chirp is determined, e.g. 35 - 45 kHz

$$\text{Lower frequency bound} = 35\text{ kHz} \times \text{mod\_value} \tag{4.4}$$

$$\text{Upper frequency bound} = 45\text{ kHz} \times \text{mod\_value} \tag{4.5}$$

Where: mod_value = 0.000032.

Mod_value is a block input parameter which produces a 1 Hz sine wave from the frequency modulator processing block provided with GNU Radio. An array of changing frequency values must be produced in order for a sample set representing the chirp to be generated. The step size of frequency values in the array is calculated using equation 4.6.

$$\text{Step size} = \frac{u-l}{s} \tag{4.6}$$

Where: u = upper frequency bound
l = lower frequency bound
s = the number of samples to generate (which impacts on the pulse length).

Entering the frequency bound and step size values into the definition for the frequency array will produce the required complex waveform from the modulator block output.

Pulse Length Definition

The number of samples created will affect the desired pulse length. This is determined by the user via adjustable registers. The necessary register values to facilitate the user's requested parameters are calculated and resolved by the program according to the USRP DAC rate. The DACs are hardwired to process samples at twice the FPGA clock frequency. The USRP onboard oscillator produces a 64 MHz clock. Therefore, the DAC processing rate is 128 M samples/sec.

This rate must be maintained to avoid sample under-runs, which may produce unwanted distortions in the transmitted signal. The FPGA Tx chain allows interpolation of a sample stream up to a factor of 512 (which is determined jointly by the default factor 4 for interpolation done by the AD9860 chips). Thus, the minimum sample rate to the Tx chains is calculated as 250 ksamples/sec using equation 4.7.

$$\text{Minimum sample rate} = \frac{\text{DAC rate}}{I} \tag{4.7}$$

Where: DAC rate = DAC sample conversion rate of 128 M samples/sec.
I = the maximum interpolation rate of 512.

The minimum rate of 256 ksamples/sec translates to 256 samples/ms. In other words, 256 samples are required for a signal duration of 1 millisecond. The test application uses an interpolation rate of 500 to produce 256 complex samples (256 I and 256 Q samples) which represent a 1 millisecond long chirp pulse.

Note that the interpolation rate set in the GNU Radio Python script is actually a combination of 2 interpolation rate values on the USRP. The AD9860 chips are set by default to interpolate at a value of 4. Therefore, the user must select an interpolation rate that is divisible by 4. The relation between the interpolation rate and the pulse length is defined as follows:

$$\text{Pulse length} = \frac{N}{\frac{32e6}{\text{interp}}}$$

Where: Pulse length is in seconds

N = the number of complex samples

interp = is the selected interpolation rate (any multiple of 4 between 4 and 512)

For example, a sample set of 256 complex samples and an interpolation rate of 200 would produce a pulse length of 1.6 milliseconds. This sample rate of 256 000 samples/sec is well above the 80 000 samples/sec minimum determined by the Nyquist Rate applied to the 40 KHz centre frequency of the chirp pulse.

But why the combination of 256 000 samples /sec and an interpolation rate of 500 to produce a 1 ms pulse? The reason is the operation of the USB 2.0 standard. For the proposed sonar application, it is convenient to produce 256 real and 256 imaginary samples which together, equate to 2 USB 2.0 burst packets.

If the sample set size did not total one USB 2.0 packet of data, then there would be samples that would be left in the transfer buffers, which would affect the size of the transmitted pulse and the operation of the system in general. Thus, if a non-matching sample set is produced, it has to be padded to ensure that the entire set is transferred.

In the diagram below, CASE 1 shows a successful sample set transfer. CASE 2 shows what happens when the size of the sample set is not matched to the USB 2.0 transfer packet size. Part of the second packet (darker shade) is left at the sender, waiting for extra data to fill the packet size.

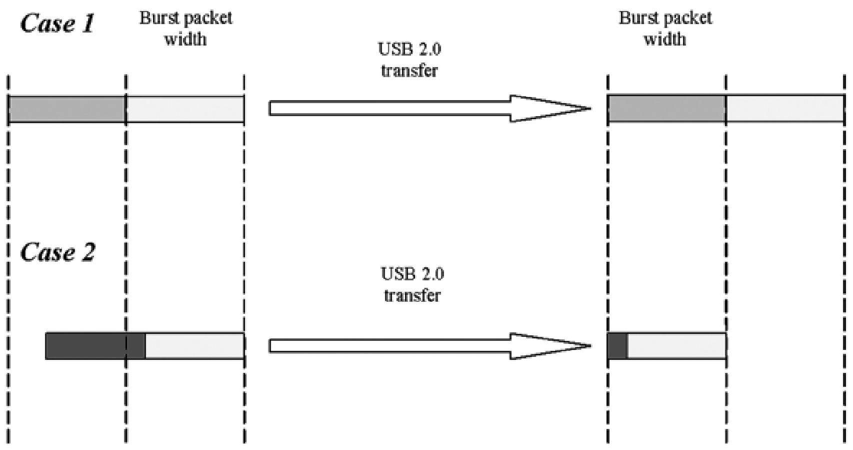


Figure 4.1: Basic diagram showing the interaction between the USB 2.0 standard and the GNU Radio sample sets.

It is possible to implement padding and obtain greater flexibility in usable sample set sizes. For the purpose of this test application however, the simpler more limited functionality is sufficient.

Once the sample set is ready, it is transferred to the USRP FPGA and stored in a 2 kB RAM module implemented in the FPGA's onboard memory. The RAM module holds the complex samples in their interleaved state. From here, the same signal can be sent continuously at periodic intervals determined by the Pulse Repetition Frequency (PRF).

Pulse Repetition Frequency Definition

Currently, the Pulse Repetition Interval (PRI) of the sonar system is 10 milliseconds. The PRI is simply the reciprocal of the PRF. Using registers to set the PRI is limited by the size of the FPGA registers, which is 32 bits. The maximum PRI is therefore 67.108863 seconds. The minimum is simply governed by the pulse length, and metrics such as range and Doppler ambiguity. [3]

Gain Definition

The signal gain can be adjusted both in the software and the hardware domain with the default system. In software, users have access to signal gain via adjusting sample amplitudes with multiplication blocks. These provide a simple way to adjust the signal amplitude. This is determined before the sample set is generated however, and as such the sample amplitude value persists once the waveform is created and it cannot be altered without generating a new sample set. Analogue gain is achieved via the programmable gain amplifier (PGA) located in the AD9860 DSP chips. These allow amplification of the signal output from the DAC by up to 20 dB.[15]

In summary, the transmitter settings the user can adjust in the sonar application are:

- The waveform (i.e. waveform type, bandwidth and centre frequency).
- The pulse repetition rate.
- The digital signal amplitude.
- The programmable gain amplifier (AD9860).

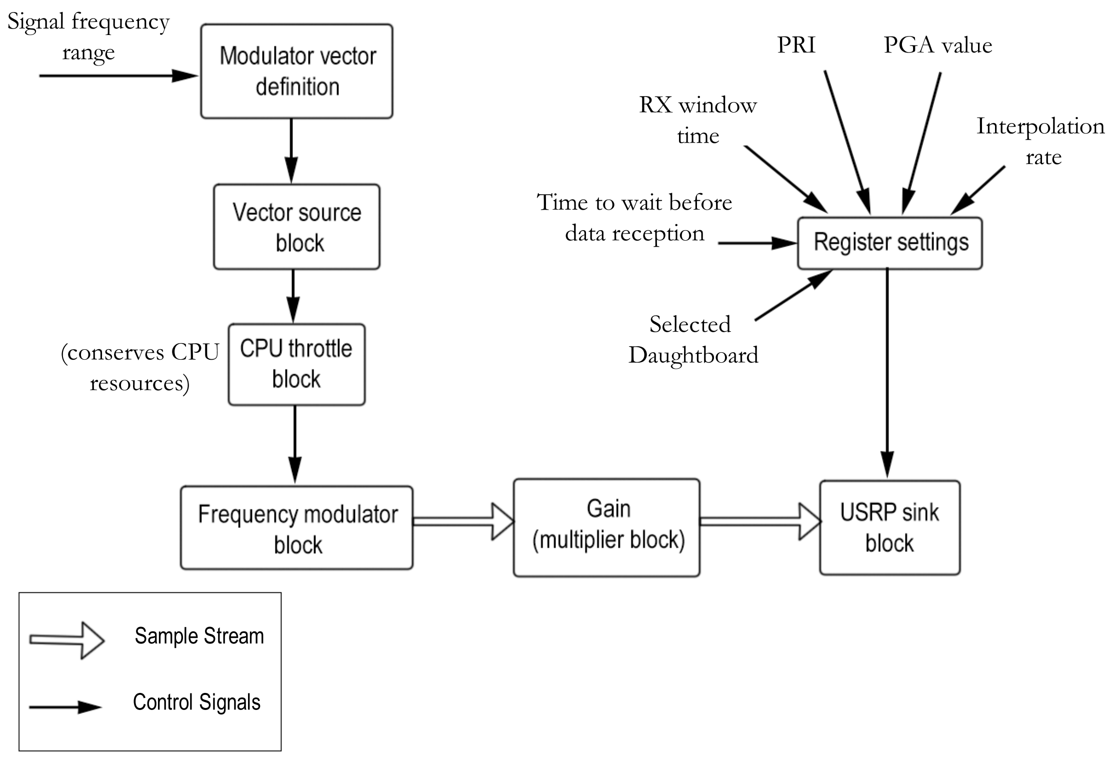


Figure 4.2: Diagram showing the sonar application setup and transmit process.

4.3.2 GNU Radio Receiver Overview

Receive Window Definition

The receive window is synonymous with the look time of the sonar. The user defines a particular time for the sonar to receive data following the PRI pulse. The GNU Radio program then resolves this requested value into a counter setting, which controls the window by regulating the number of samples that cross the USB link during reception. These received sample windows are relative to the PRI trigger pulse.

Digitisation Delay Definition

The user is able to define a digitisation delay period after the PRI pulse. This allows the receiver to be blinded for a particular near-field range, which is particularly useful for radar. This delay in combination with controllable hardware front-ends such as the FLEX series daughterboards, protects the receiver from the high energy of the transmitting pulse, and prevents the ADCs from saturating. This also allows the signal processing to be focused on particular ranges. This can ease processing by reducing the resultant overall data rate.

Like the receiver window parameter, the digitisation delay is set as a finite time value by the user. The GNU Radio program resolves this time into a number of samples to discard before activating transfer over the USB link.

The reason for this implementation is that the FPGA cannot control the activation or deactivation of the ADCs within the AD9860 chips. Thus, the sample flow must be regulated within the USRP FPGA. With other daughterboards such as the RFX series, it becomes possible to control the signal flow before the ADCs, which would protect them from saturating. This functionality is not available with the Basic Rx board used for this particular project. It is not necessary however, due to the robustness of the front-end hardware being used, and the relatively low power signals being transmitted and received.

Data Capture and Processing

The current implementation of this sonar application uses non-real time data processing. The GNU Radio script is configured to gather received complex samples into a binary file for processing in Matlab.

Upon launching the program, a graphical plotting window is generated which shows received signal activity. In addition, console outputs are used to give debug messages to show the stage of operation the program is executing. Among the real time plotting options for viewing the raw received data are an oscilloscope type plot indicating signal levels, and an FFT plot to show the frequency content of the received data.

The received data is finally processed in Matlab. The processing done includes match filtering using a waveform sample set generated by GNU Radio, correlated with the received data. The tests that were performed and their results will be presented in the next chapter.

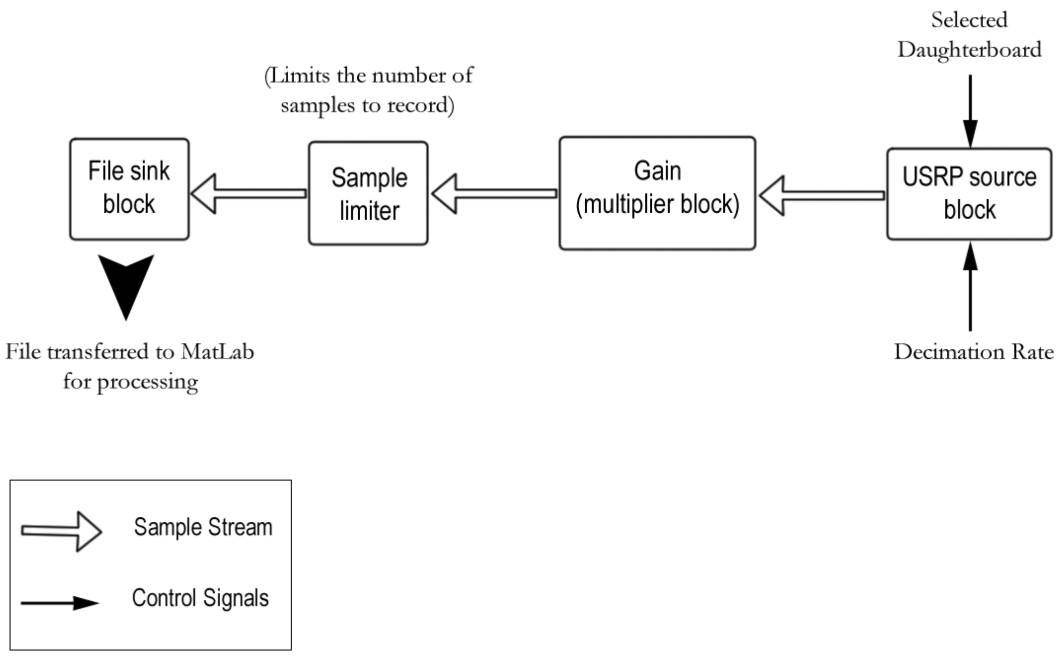


Figure 4.3: Diagram showing the sonar receiver process.

4.4 FPGA / Verilog Modifications

As with the GNU Radio source files, the Hardware Description Language (HDL) files for the USRP FPGA are also Open Source. This allows designers to examine the inner workings of the FPGA and to make modifications. Every time a GNU Radio application which uses the USRP is launched, the GNU Radio subsystem will program the FPGA using a pre-compiled binary file. Either the default binary file is loaded, or another custom file of the user's choosing (set within the GNU Radio program).

Custom binary files are compiled from Verilog source files in Altera's Quartus program[20]. The default Quartus project directories used by the USRP are packaged along with the GNU Radio source files. Developers can use this as a basis for modifications. The custom FPGA binary files used for this thesis were developed by using the free web edition of Quartus. They can be viewed on the attached compact disk, the file directory of which can be seen in Appendix C.

The following sections will fully describe the modifications that had to be made to the default USRP FPGA configuration. The changes will be explained in terms of the transmit functionality and the receive functionality.

4.4.1 FPGA Transmit Chain Modifications

The PRF is generated by a counter which counts down from a register value determined by the user. On the positive edge of the PRF signal, the transmission of a chirp pulse begins. The PRI is determined in clock cycles.

The Tx buffer module has also been modified. It was decided to partially retain this module because it contains the basis for bridging the FPGA and Cypress FX2 chip clock domains. Where normally this module would both adjust the samples rates and do de-interleaving, it is now just used to buffer and transfer the samples that must be transmitted. The buffering is done in a First-in-First-Out (FIFO) queue, which does consume memory space that might be otherwise utilised. The de-interleaving process has been moved to the RAM control module, which will be explained below.

The RAM control module contains the counters governing the PRF, the receiver window, the digitisation delay, and the RAM module access. This control module also handles de-interleaving data from the USB link. Because this system only implements one complex channel, the de-interleaving process only splits the I-Q sample streams to their respective Tx chain modules.

Samples forming the pulse to be transmitted are stored in a RAM module implemented in the FPGA. The RAM module is larger than the size required to store the transmit waveform. This is to make it functional for longer pulse lengths which would require more signal samples. From the waveform stored in the RAM, the same pulse can be triggered continuously. Ideally the user should be able to control the number of samples that forms the signal to be transmitted, thereby controlling the length of the transmitted waveform.

As the sample stream leaves the RAM module, it is sent to the Tx chain module. A cascaded integrator comb filter is used to interpolate the sample stream. The rate that samples enter this stage is controlled by an interpolation strobe generated in another part of the FPGA. This rate is calculated by the GNU Radio software using the interpolation rate that the user requests. The samples are then sent to the AD9860 DSP chip. The Verilog code for the Signal Control Module, depicted in the diagram below, can be viewed in Appendix B.

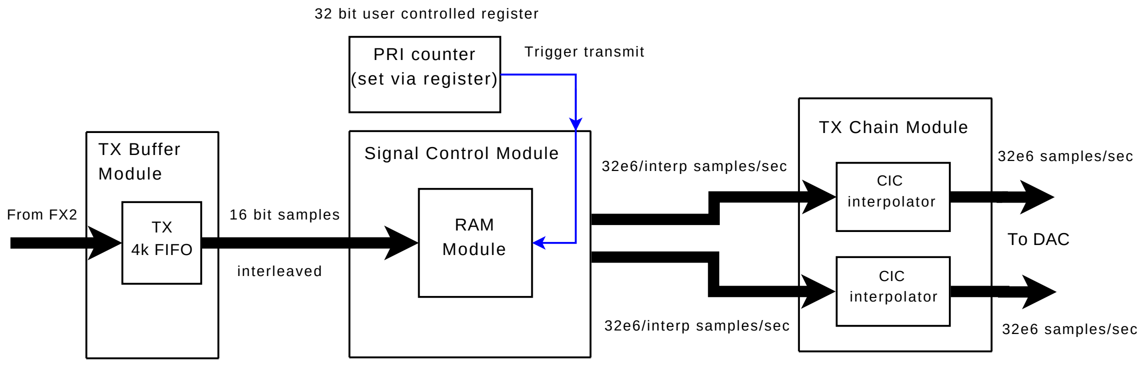


Figure 4.4: Diagram showing a system level view of the modified FPGA transmit functionality.

4.4.2 FPGA Receive Chain Modifications

In the FPGA, most of the receiver modules are left unchanged. It was decided that the best place to control the data flow was just before the USB buffer stage. The ADC offset correction and Rx chain modules are left to run continuously. The NCO in the Rx module is disabled as the signal is being received and processed at baseband.

Because the ADC sample rate cannot be dynamically controlled, the full 64 MHz of data is being collected, even though only 2 kHz of bandwidth is required. The CIC decimators are set to reduce the sample rate to 256 ksamples/second. From here, the sample stream is connected to the USB buffer module.

In the buffer module control signals are gathered from the transmitting RAM module. When the PRF signal triggers transmission, then the delay counter activates. The Rx window counter initiates once this counter expires. Upon the PRF signal occurring, full amplitude scale samples are inserted into the receiver data stream travelling across the USB link. The beginning of these marker samples indicates when the PRI signal occurred, relative to the receiver data that is about to arrive.

Once the delay counter expires and the Rx window counter begins, the receiver data stream is re-linked to the ADC output. The samples that arrive from the ADC after the decimation stage are counted by the Rx window counter.

Once the number of received samples equals that of the user determined Rx window size, then the ADC data link to the USB buffer module is interrupted. Samples from the ADCs are no longer routed to the host computer after the Rx window counter expires. The receiver process is repeated at the next PRI signal.

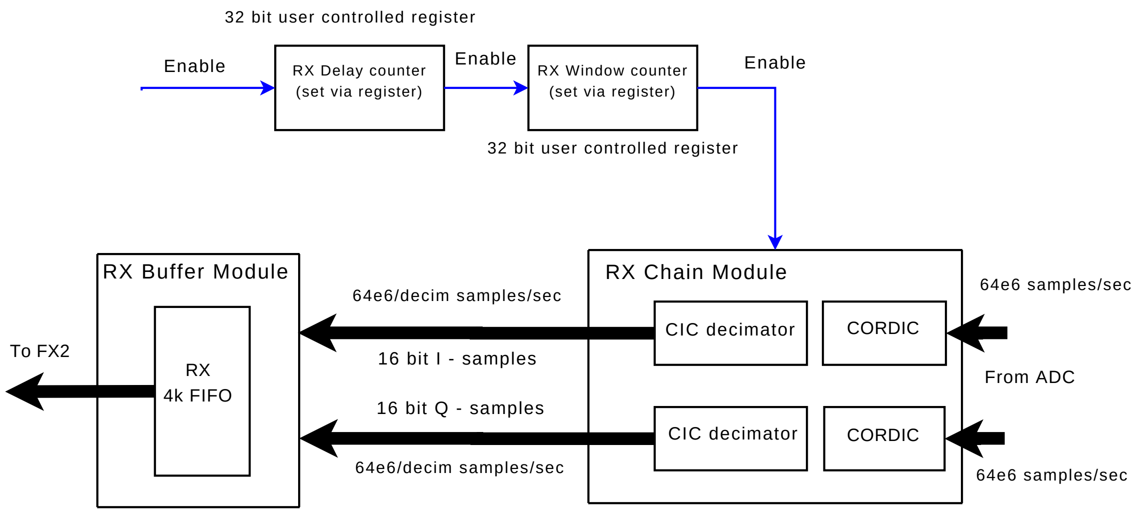


Figure 4.5: Diagram showing a system level view of the modified receive functionality within the FPGA.

4.5 Daughterboards

4.5.1 Transmit Daughterboard

The daughterboard being used in the application is the Basic Tx board. It contains no analogue hardware other than an output transformer and a SMA connector. The output from this board is connected to a small analogue front-end which provides some gain and voltage level adjustment to drive the ultrasonic transducers.

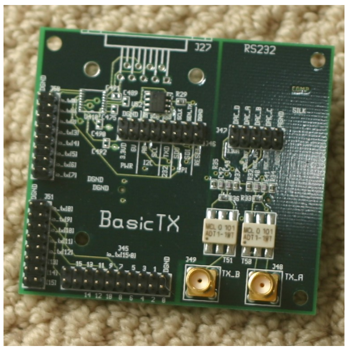


Figure 4.6: Picture of a Basic Tx daughterboard for the USRP.

4.5.2 Receive Daughterboard

On the receiver side, the basic Rx daughterboard is used. The sonar centre frequency is low enough for no additional mixing hardware to be required.

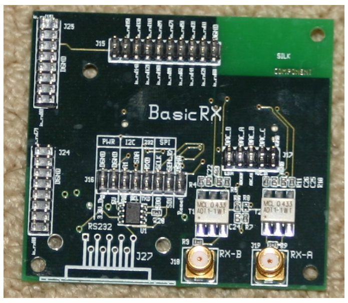


Figure 4.7: Picture of a Basic Rx daughterboard for the USRP.

4.6 Front-end Hardware

4.6.1 Transmitter Circuitry

The transmitter circuit posed some challenge in development for producing a suitably amplified voltage to drive the transducers. This was also important to provide as much transmitted pulse power as possible, since the pulse length would be short for an acoustic wave. It was also found that inexpensive opamps such as the LF347 [21] could not provide enough driving voltage for a pulsed signal to be detectable over any significant range (beyond 10 cm). Such opamps did have sufficient Gain Bandwidth Product (GBWP) values for operation at 40 kHz, which made them ideal for circuit filters.

Initially, a pre-built driver circuit constructed for another project similar to the system described in [22, 23, 24] was tested. It utilised switched capacitor filters, and the LM3886 [25] audio amplifier. While the driver circuit was effective, it was cumbersome and contained much circuitry that was unnecessary for this project. Many components used (such as filters and mixers) may have contributed signal interference due to the overall circuit construction and layout.

While attempting to use a less expensive and less power consuming amplifier, a RF transformer was designed in order to maximise the transducer driving voltage. Using design notes from [26, 27], a RM core was selected as it could achieve high inductance at ultrasonic frequencies with a relatively low number of turns. [28, 29] The transformer was wound to match the initial capacitance of the ultrasonic transducer before it reached resonance.

A turn ratio of 1:10 was selected in order to keep current demands on the driving opamp manageable. This would allow the cheaper LM380 [30] audio amplifier to be used. The circuit built was based on design notes in [30].

It was found however, that the use of a transformer resulted in distortions in the transmitted pulse over a significant portion of its duration. This was related to the rapidly forming and collapsing magnetic field in the transformer core due to the short pulsed transmission required.

To minimise the distortion, a more rigorous transformer design would be required, which was beyond the scope of this project. In testing it was found that the transformer would be a more viable option for continuous wave (CW) transmission applications, or a pulsed application using much longer pulse durations (greater than 10 ms), where the finite time distortions would be more tolerable.

A much simpler driver circuit was developed that was modified from [22, 23]. This utilised a simple opamp based bandpass filter depicted below.

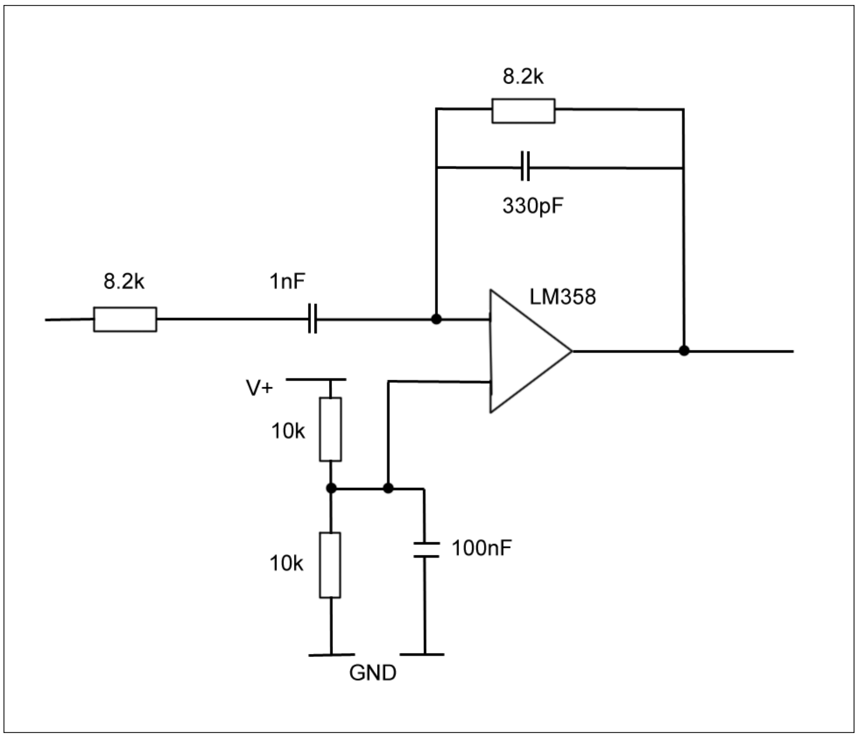


Figure 4.8: Circuit diagram of transmitter circuit filter.

This filter utilised the relatively inexpensive LM358. The filter's passband was calculated as 19.4 - 58.8 kHz using equations 4.8 and 4.9.

$$\text{Low frequency breakpoint} = \frac{1}{2\Pi \times R1 \times C1} \tag{4.8}$$

$$\text{High frequency breakpoint} = \frac{1}{2\Pi \times R2 \times C2} \tag{4.9}$$

The gain of this stage was kept to unity so as not to strain the opamp's GBWP. The final driving stage was built around the LM3886 with a gain of 56. The circuit is depicted below.

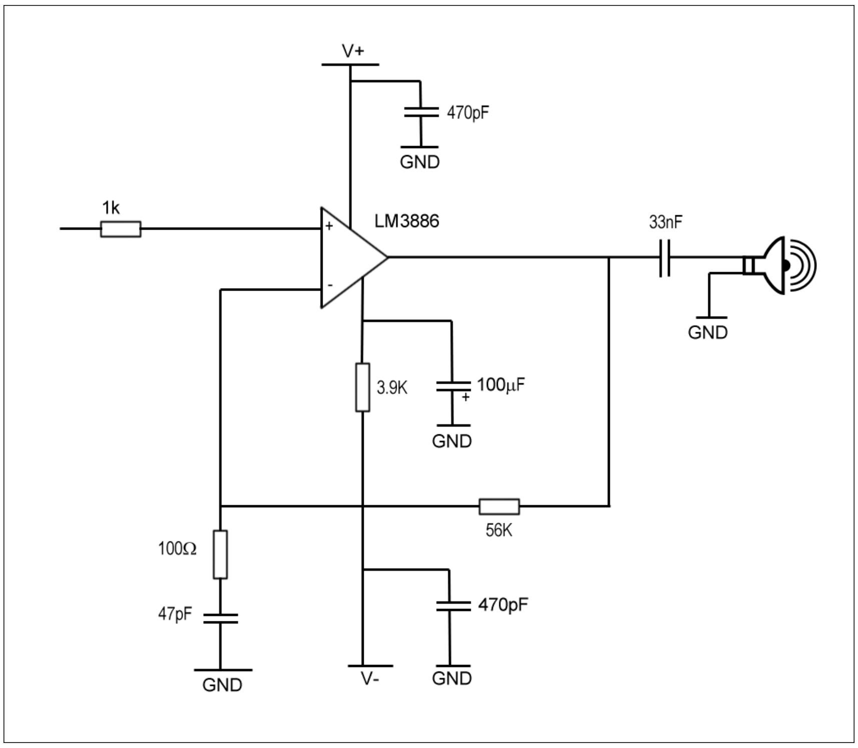


Figure 4.9: Circuit diagram of the Tx transducer driving circuit.

4.6.2 Receiver Circuitry

The receiver circuit is derived from the Murata ultrasonic transducer datasheet [31] as the suggested circuit for receiving and amplifying signals from an ultrasonic transducer. A single supply LF347 circuit is used instead of the recommended opamp. The circuit combines amplification by and bandpass filtering.

The two transducers are positioned to face the same area. They are linked to the front-end circuitry with shielded cables to minimise electrical signal coupling between the transmitter and receiver circuitry.

4.7 Prototype Development Summary

An overall system view of the sonar application is depicted in figure 4.10.

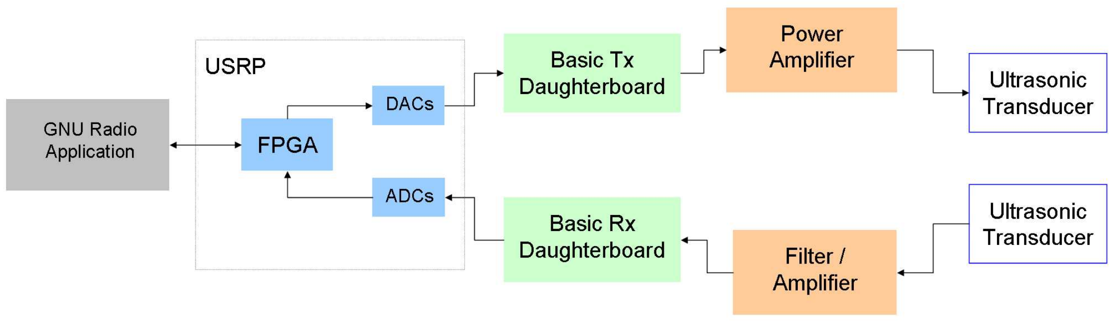


Figure 4.10: System level diagram of the USRP based sonar application.

The FPGA modifications were completed and were able to perform to the specifications listed in 4.1. Fully variable pulse durations by changing the sample size could not be implemented. Varying the pulse duration is possible through the interpolation rate, as will be shown in 5. As testing revealed however, this method of pulse duration control produces undesirable effects and should not be used.

The following functionality was finally achieved:

- User-defined waveform generation using GNU Radio processing blocks.
- Successful waveform sample set transfer to, and storage within the FPGA.
- Accurate transmission of the waveform stored in the FPGA.
- Fully adjustable PRI controller.
- Accurate triggering of the transmission process governed by the PRI.
- Adjustable digitisation delay.
- Fully adjustable receive window.
- Accurate reception of received data transferred to the host computer from the USRP.
- Successful gathering and archiving of received data using GNU Radio processing blocks.
- Development of driver and filter circuitry that was able to adequately perform for sonar testing purposes in a lab environment.

The following chapter will describe the system tests performed and the sonar experiments that were done with this USRP based monostatic sonar system.

Chapter 5

Tests and Results

The following chapter covers the tests that were performed with the USRP, GNU Radio and the sonar application that was developed. The purpose of the first few tests is to verify the existing functionality which GNU Radio and the USRP include by default and then go on to basic tests of the functionality that was created and incorporated as part of this thesis project. Later tests will further depict the capability of the developed functions through using the sonar to detect a target. The system's ability to capture useful received data accurately will be proved. The captured data will then be processed to show that the range of the detected target from the sonar transmitter can be correctly calculated.

In all tests with the receiver subsystem, only the real channel was gathered from the front-end circuitry. The Basic RX boards do not contain I-Q phase shifting, and this functionality was omitted from the constructed front-end circuitry for simplicity. In addition, even though the low system bandwidth, low environmental acoustic noise and short pulse periods negated the need for chirp waveform tests, these were none the less carried out to show that such waveforms could be readily produced, and that the receiver subsystem could recover them.

The test results and comments on these results are included.

5.1 System Performance vs. Specifications Tests

Purpose:

To confirm specified and expected performance limits of the system's software-defined transmitter sections, and to demonstrate the gain control available on the USRP.

Description:

The USRP Tx output is connected directly to an oscilloscope input. Chirp pulses are then emitted with varying application parameters. The transmitted pulses with different settings are recorded with the oscilloscope. The parameters which are altered are: software gain, PGA gain, PRI and pulse length.

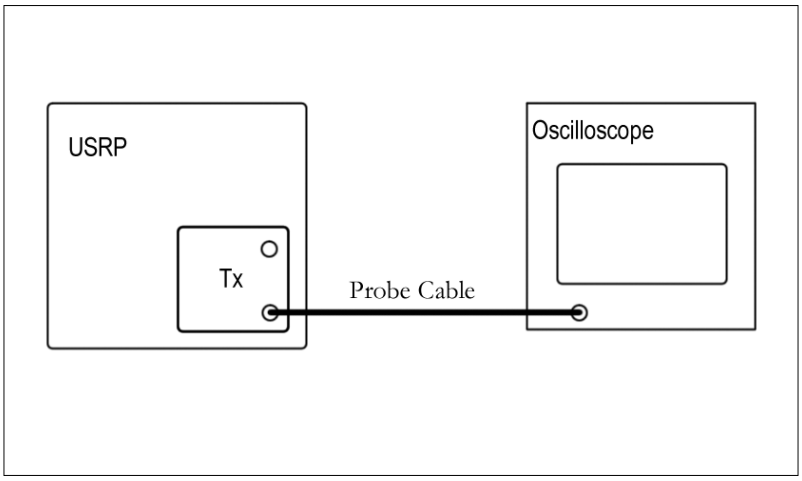
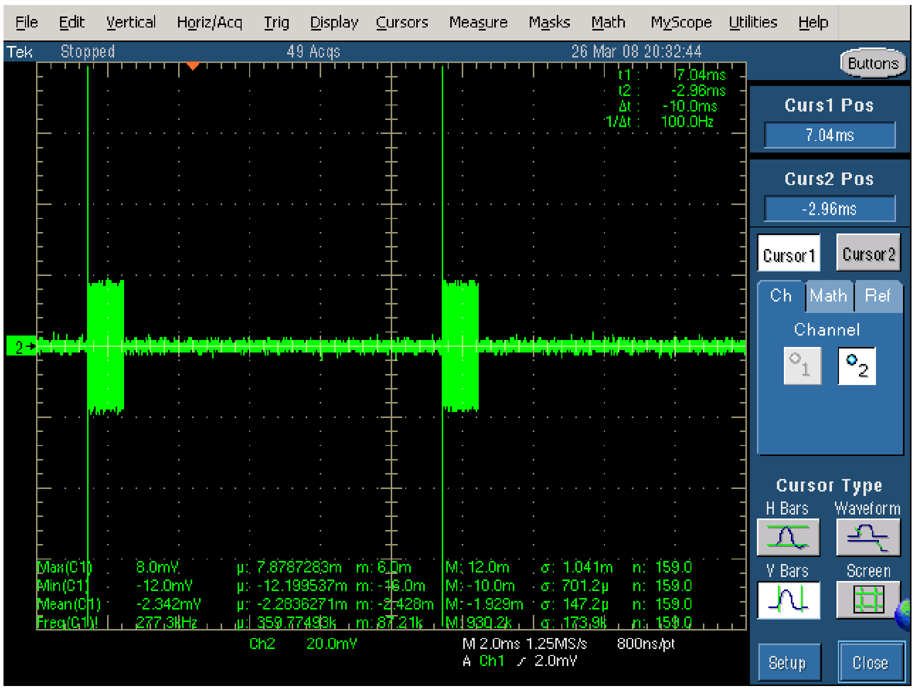


Figure 5.1: Diagram depicting performance versus specifications test setup.

5.1.1 Software Gain Test

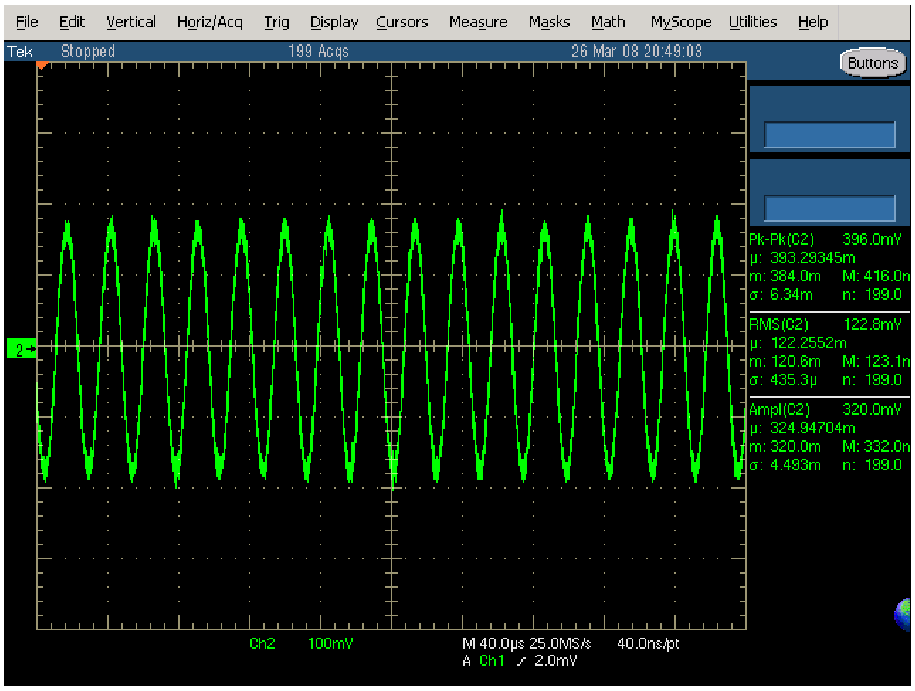
The objective of the software gain test is to demonstrate the control in sample amplitude available from GNU Radio for the sonar application, considering the daughterboards being employed.

Settings:	Interpolation rate = 500
	t_{pri} = 10 ms
	τ = 1 ms
	Δf = 39-41 kHz
	PGA gain = 20 dB



```
Tek   Stopped          49 Acqs             26 Mar 08 20:32:44

  t1 :    7.04ms
  t2 :   -2.96ms
  Δt :   -10.0ms
1/Δt :   100.0Hz

Curs1 Pos  7.04ms
Curs2 Pos  -2.96ms

Max(C1)    8.0mV     µ: 7.8787283m   m: 6.0m     M: 12.0m    σ: 1.041m   n: 159.0
Min(C1)    -12.0mV   µ: -12.199537m  m: -16.0m   M: -10.0m   σ: 701.2µ   n: 159.0
Mean(C1)   -2.342mV  µ: -2.2836271m  m: -2.428m  M: -1.929m  σ: 147.2µ   n: 159.0
Freq(C1)!  277.3kHz  µ: 359.77493k   m: 87.21k   M: 930.2k   σ: 173.9k   n: 159.0

          Ch2     20.0mV              M 2.0ms 1.25MS/s    800ns/pt
                                      A  Ch1  ⸝  2.0mV
```

Figure 5.2: Trace showing software gain reduced to 4000. V_{pp} = 46mV.



```
Tek   Stopped          199 Acqs            26 Mar 08 20:49:03

Pk-Pk(C2)      396.0mV
µ:  393.29345m
m: 384.0m      M: 416.0n
σ:  6.34m      n:  199.0

RMS(C2)        122.8mV
µ:  122.2552m
m: 120.6m      M: 123.1n
σ:  435.3µ     n:  199.0

Ampl(C2)       320.0mV
µ:  324.94704m
m: 320.0m      M: 332.0n
σ:  4.493m     n:  199.0

          Ch2     100mV               M 40.0µs 25.0MS/s    40.0ns/pt
                                      A  Ch1  ⸝  2.0mV
```

Figure 5.3: Trace showing maximum software gain. Setting = 32767. V_{pp} = 364 mV.

Test Summary

The average step size for software gain to output voltage is approximately 11.2 μV. Documentation indicates that the AD9860 chips are capable of delivering 2 V_{pp} to a 50 ohm load. The reason this value could not be attained could possibly be attributed to reduction in bit gain through the FPGA implemented CIC interpolator. It was also noted that at the sonar operating frequency, the output transformers of the basic daughterboards were expected to attenuate the signal. The low frequency breakpoint was documented at 100 kHz. The signal amplitude was still sufficient however for the purposes for which the sonar was used. Reducing the software gain effectively mutes the output.

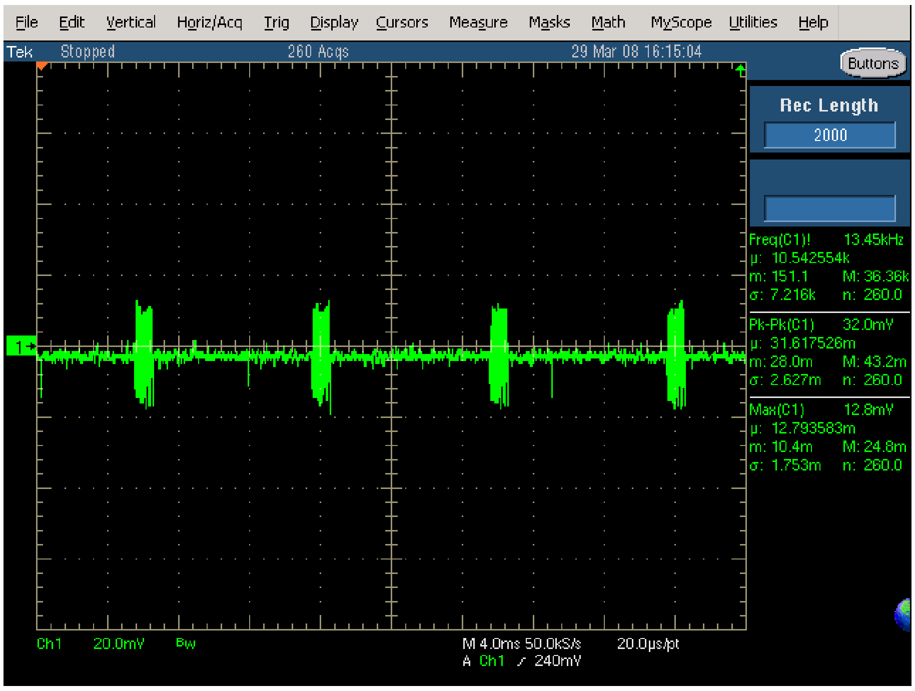
5.1.2 PGA Gain Test

The objective of this test was to demonstrate the gain control provided by the user controlled PGA in the AD9860 chip on the USRP.

The PGA value range for the daughterboard plugged into the USRP can reported within GNU Radio via a program class method. The reported value range for the Basic Tx board is -20 to 0 in step sizes of 0.0784313. The default setting is maximum PGA gain of 20 dB (represented by value 0) while the minimum is 0 dB gain (indicated by value -20). The user selects the PGA gain as a dB value.

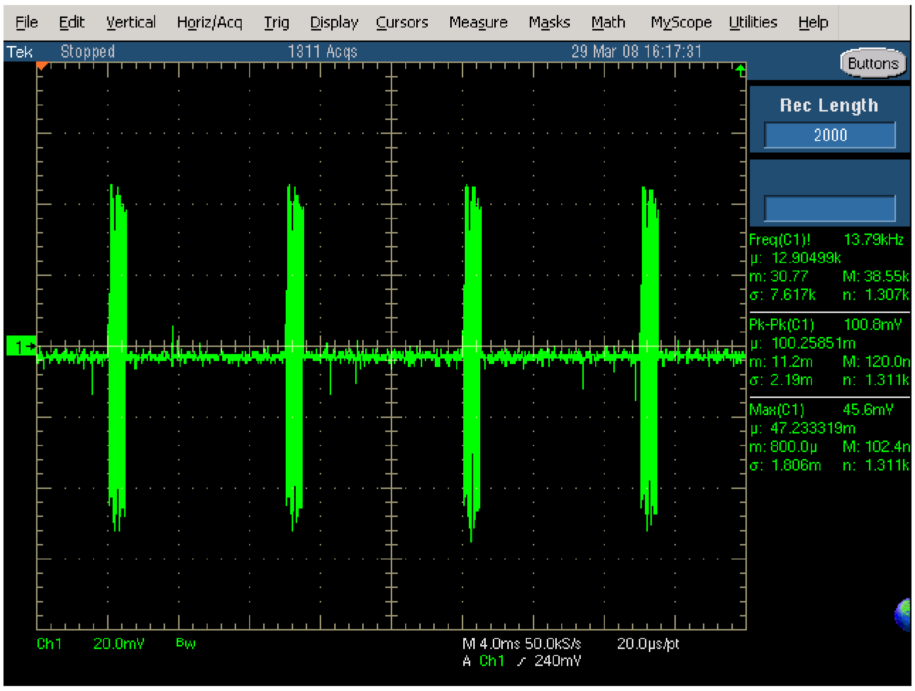
Settings: Interpolation rate = 500
t_{pri} = 10 ms
τ = 1 ms
Δf = 39-41 kHz
SW gain = 32e6

Setting the PGA to its maximum of 20 dB gain with software gain set to 32e6 yielded 356mV as depicted in figure 5.3.



```
Tek   Stopped          260 Acqs             29 Mar 08 16:15:04
Rec Length
2000
Freq(C1)!      13.45kHz
µ: 10.542554k
m: 151.1       M: 36.36k
σ: 7.216k      n: 260.0
Pk-Pk(C1)      32.0mV
µ: 31.617526m
m: 28.0m       M: 43.2m
σ: 2.627m      n: 260.0
Max(C1)        12.8mV
µ: 12.793583m
m: 10.4m       M: 24.8m
σ: 1.753m      n: 260.0
Ch1    20.0mV    Bw          M 4.0ms 50.0kS/s      20.0µs/pt
                             A  Ch1  ⟋  240mV
```

Figure 5.4: Trace showing output when PGA is set to 0 dB (minimum). V_{pp} = 32 mV.



```
Tek   Stopped          1311 Acqs            29 Mar 08 16:17:31
Rec Length
2000
Freq(C1)!      13.79kHz
µ: 12.90499k
m: 30.77       M: 38.55k
σ: 7.617k      n: 1.307k
Pk-Pk(C1)      100.8mV
µ: 100.25851m
m: 11.2m       M: 120.0n
σ: 2.19m       n: 1.311k
Max(C1)        45.6mV
µ: 47.233319m
m: 800.0µ      M: 102.4n
σ: 1.806m      n: 1.311k
Ch1    20.0mV    Bw          M 4.0ms 50.0kS/s      20.0µs/pt
                             A  Ch1  ⟋  240mV
```

Figure 5.5: Trace showing output when PGA is set to 10 dB.V_{pp}= 100 mV.

Test Summary

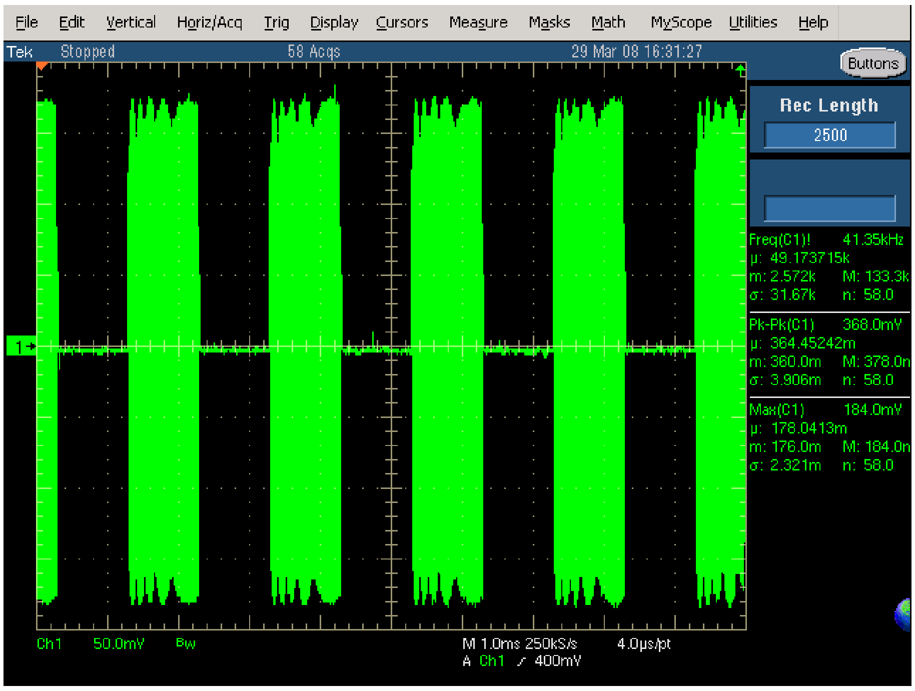
The PGA behaved as expected. For the tests following in the chapter, the values of software gain and PGA gain were kept at their maximum settings.

5.1.3 PRI Settings Test

The objective of this test is to demonstrate the functionality and expected limits of the PRI function added to the FPGA.

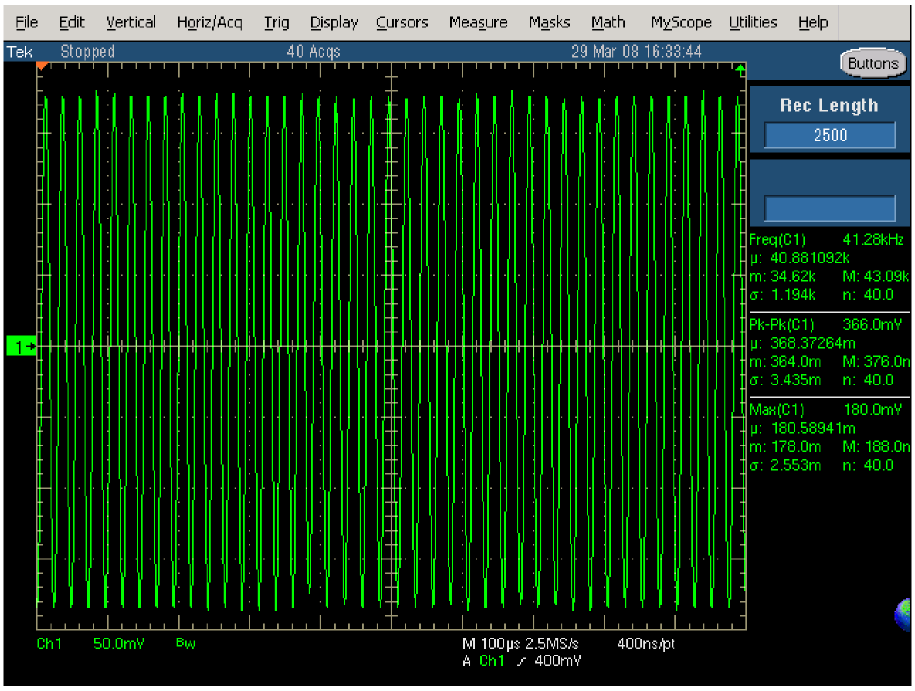
The PRI counter within the FPGA is a standard counter module and is able to operate as such without restriction other than the capabilities of the FPGA itself. For the purposes of radar and sonar, a practical limit is set by the transmitted pulse duration. The maximum possible value is limited by the width of the counter as described in section 4.3.1.

Settings:	Interpolation rate = 500
	τ = 1 ms
	Δf = 39-41 kHz
	SW gain = 32e6
	PGA = 20 dB



```
Tek   Stopped   58 Acqs   29 Mar 08 16:31:27

Rec Length
2500

Freq(C1)!     41.35kHz
μ: 49.173715k
m: 2.572k     M: 133.3k
σ: 31.67k     n: 58.0

Pk-Pk(C1)     368.0mV
μ: 364.45242m
m: 360.0m     M: 378.0m
σ: 3.906m     n: 58.0

Max(C1)       184.0mV
μ: 178.0413m
m: 176.0m     M: 184.0m
σ: 2.321m     n: 58.0

Ch1   50.0mV   Bw       M 1.0ms 250kS/s   4.0μs/pt
                        A Ch1 ⟋ 400mV
```

Figure 5.6: Trace showing t_{pri} = 2 ms (2τ).



```
Tek   Stopped          40 Acqs          29 Mar 08 16:33:44

Rec Length  2500

Freq(C1)      41.28kHz
μ:  40.881092k
m: 34.62k     M: 43.09k
σ:  1.194k    n:  40.0

Pk-Pk(C1)     366.0mV
μ:  368.37264m
m: 364.0m     M: 376.0m
σ: 3.435m     n:  40.0

Max(C1)       180.0mV
μ:  180.58941m
m: 178.0m     M: 188.0m
σ: 2.553m     n:  40.0

Ch1   50.0mV   Bw        M 100μs 2.5MS/s    400ns/pt
                         A  Ch1  ⁄  400mV
```

Figure 5.7: Trace showing t_{pri} = 1 ms. PRI = τ, thus the output waveform has become continuous.

Test Summary

The tests showed that the PRI trigger generation in the FPGA functions as intended. As shown in 5.7, reducing the PRI to match the pulse duration results in a continuous wave output. Recording the maximum possible PRI for display purposes proved difficult with the measuring equipment used. However, tests with PRI of up to 1 second still showed accurate trigger pulse timing. Using such long PRIs was not required for the testing to follow in this chapter however, the functionality to produce long PRIs does exist.

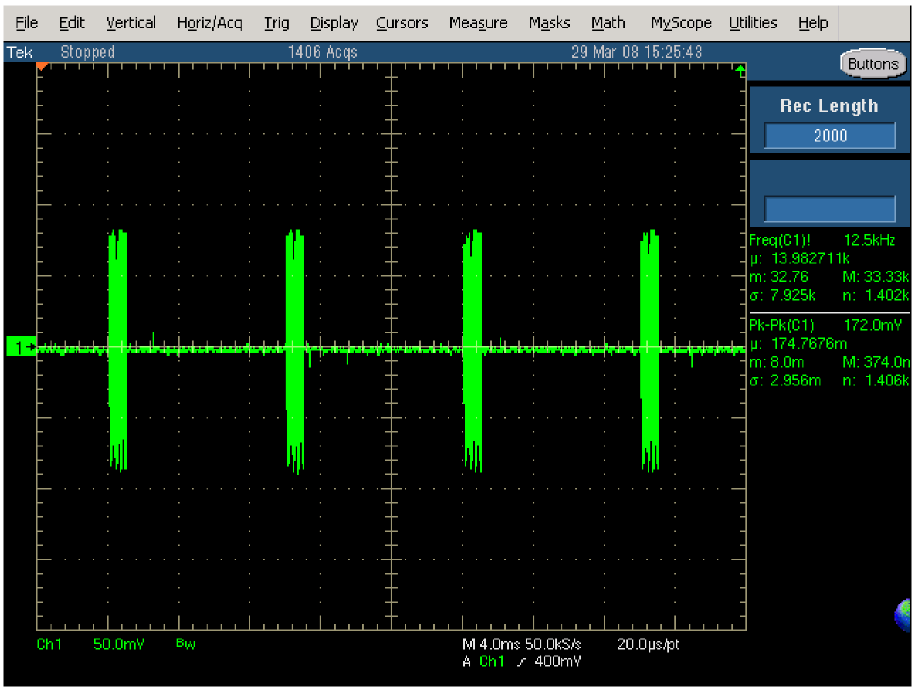
5.1.4 Pulse Length Test

The objective of this test is to show the limits of the control of the transmitted pulse duration. The correct functionality of the pulse duration control via variant samples lengths is shown. The maximum capacity of the RAM implemented on the FPGA could not be tested. In addition, the zero padding features of the transferred sample sets were not yet functional in the tested system. As a result, the tested system was limited to 1 ms and 500 μs pulse durations. These two periods are determined by the sample set size which is impacted by the USB standard as described in 4.3.1. Ideally, variable pulse lengths would be attainable by forming sample sets of varying lengths.

Variable sample set size:

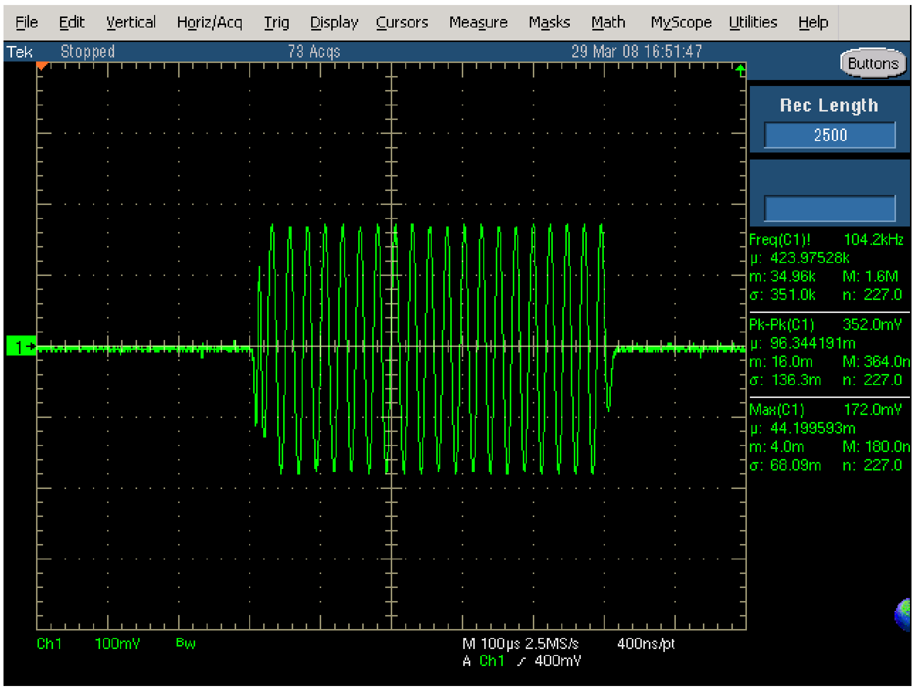
In this test, the size of the sample set transferred to the USRP's FPGA is varied. The expected effect on the output pulse duration is depicted.

Settings: Interpolation rate = 500
Δf = 39-41 kHz
SW gain = 32e6
PGA = 20 dB
t_{pri} = 10 ms



```
Tek   Stopped        1406 Acqs        29 Mar 08 15:25:43

Rec Length
2000

Freq(C1)!       12.5kHz
µ: 13.982711k
m: 32.76        M: 33.33k
σ: 7.925k       n: 1.402k

Pk-Pk(C1)       172.0mV
µ: 174.7676m
m: 8.0m         M: 374.0m
σ: 2.956m       n: 1.406k

Ch1   50.0mV   Bw        M 4.0ms 50.0kS/s    20.0µs/pt
                         A  Ch1  ⁄  400mV
```

Figure 5.8: Trace showing τ = 1 ms (512 complex samples).


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Buttons

```
Tek   Stopped        73 Acqs        29 Mar 08 16:51:47
Rec Length
2500
Freq(C1)!     104.2kHz
µ: 423.97528k
m: 34.96k     M: 1.6M
σ: 351.0k     n: 227.0
Pk-Pk(C1)     352.0mV
µ: 96.344191m
m: 16.0m      M: 364.0n
σ: 136.3m     n: 227.0
Max(C1)       172.0mV
µ: 44.199593m
m: 4.0m       M: 180.0n
σ: 68.09m     n: 227.0
Ch1   100mV   Bw        M 100µs 2.5MS/s    400ns/pt
                        A  Ch1  ↗ 400mV
```

Figure 5.9: Trace showing τ = 500 μs (256 complex samples).

Test Summary

Altering the pulse duration by changing the sample set size performed as expected. Altering the interpolation rate also changes the pulse duration as expected. It was found however that this functionality caused undesirable distortions (such as changes in centre frequency and pulse shape) in the output signal. It is recommended to only alter pulse lengths via variable sample set sizes and to determine the interpolation rate according to the desired pulse length.

5.2 Transmitter/Receiver Direct Coupling Tests

Purpose:

To determine how severe the direct signal coupling between the transmitting and receiving transducer is in the physically constructed system. Using the information from this test, the direct coupling should then be minimised. This is primarily achieved via various kinds of physical screening placed between the transducers.

Description:

Chirp pulses are transmitted into an unobstructed area where no echoes may occur. The terminals of the receiver transducer are monitored with an oscilloscope. Transmitted signals that are directly coupled into the receiver are recorded.

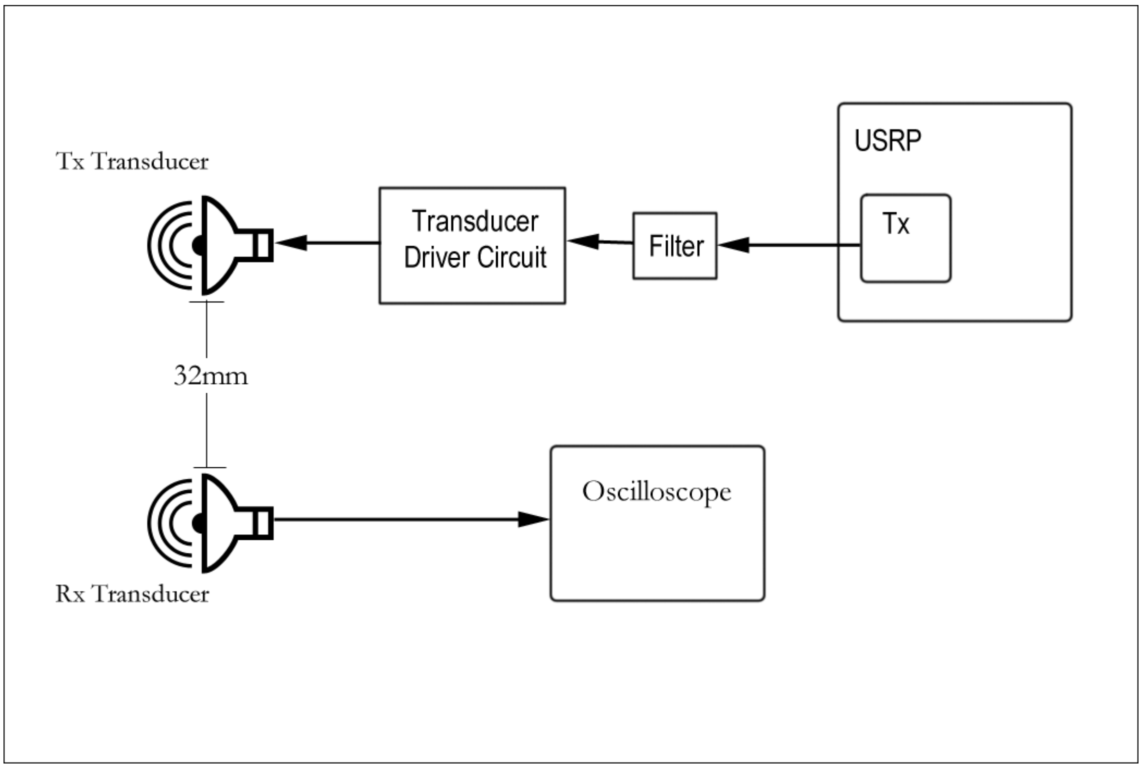
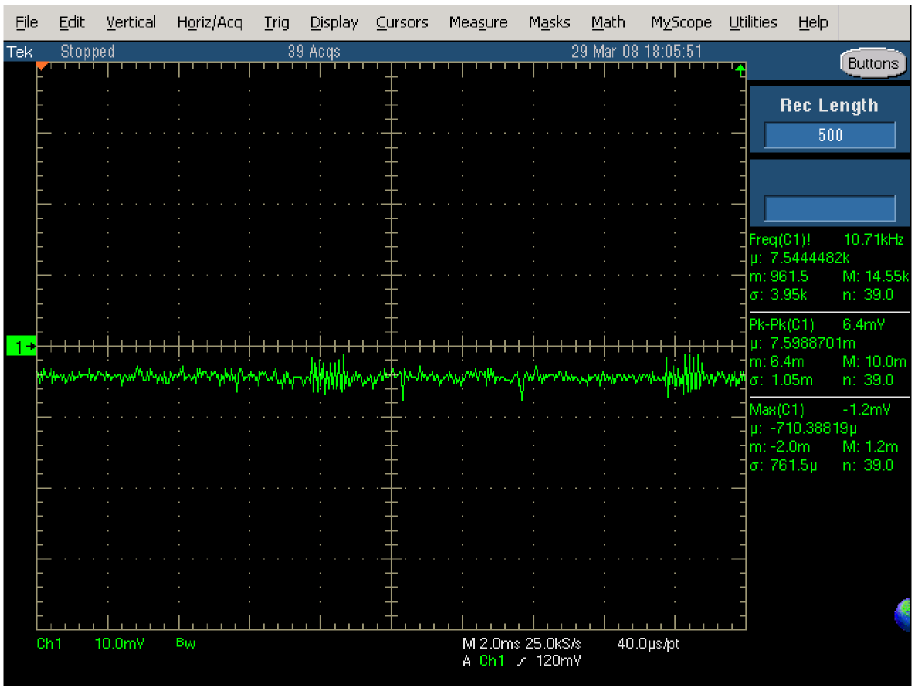


Figure 5.10: Diagram depicting transmitter/receiver direct coupling test setup.

The items used to attempt to isolate the transmitter and receiver transducers included a solid wooden shape placed between them and a similarly shaped construction made of foam. Various simple orientations of these items between the transducers was attempted.

Settings:	Interpolation rate = 500
	SW gain = 32e6
	PGA = 20 dB
	Δf = 39-41 kHz



```
Tek   Stopped          39 Acqs          29 Mar 08 18:05:51

Rec Length
500

Freq(C1)!       10.71kHz
μ: 7.5444482k
m: 961.5        M: 14.55k
σ: 3.95k        n: 39.0

Pk-Pk(C1)       6.4mV
μ: 7.5988701m
m: 6.4m         M: 10.0m
σ: 1.05m        n: 39.0

Max(C1)         -1.2mV
μ: -710.38819μ
m: -2.0m        M: 1.2m
σ: 761.5μ       n: 39.0

Ch1    10.0mV    Bw          M 2.0ms 25.0kS/s     40.0μs/pt
                             A  Ch1  ⁄  120mV
```

Figure 5.11: Trace showing the directly coupled signal detected at the receiver.

Test Summary

In the image, the directly coupled signal can be seen, having a signal amplitude of approximately 4 mV_{pp} .

A separation of 32 mm between the transducers and shielded wires appeared to minimise detectable coupling. Screening with different materials did not reduce the visible coupled signal. Even completely physically obstructing the receiving transducer from receiving any signal did not reduce the signal depicted in 5.11.This suggests that the signal is electrically coupled. Thus, it may be reduced with more careful circuit layout. For the purposes of the tests that would be performed with the developed sonar, the existence and magnitude of the detected coupled signal was tolerable.

5.3 Receiver Circuit Tests

Purpose:

Determine the echo shape of a flat plate and a trihedral target in the analogue domain. This is simply to provide some reference for visual comparison with the sampled received signal in order to assess the digital receiver's fidelity. This test is also to provide a simple means of determining which of the two target types (the flat plate or the trihedral) was more suitable for the range tests. This would be assessed by which target was easier to accurately erect, and which provided the strongest and visually clearest reflected signal.

Description:

Chirp signals are transmitted against a flat plate target, and the analogue reflected signal is recorded with a Tektronix TDS5052B digital phosphor oscilloscope. This is repeated for a trihedral corner reflector. The targets are positioned at various ranges orthogonally to the sonar transducers.

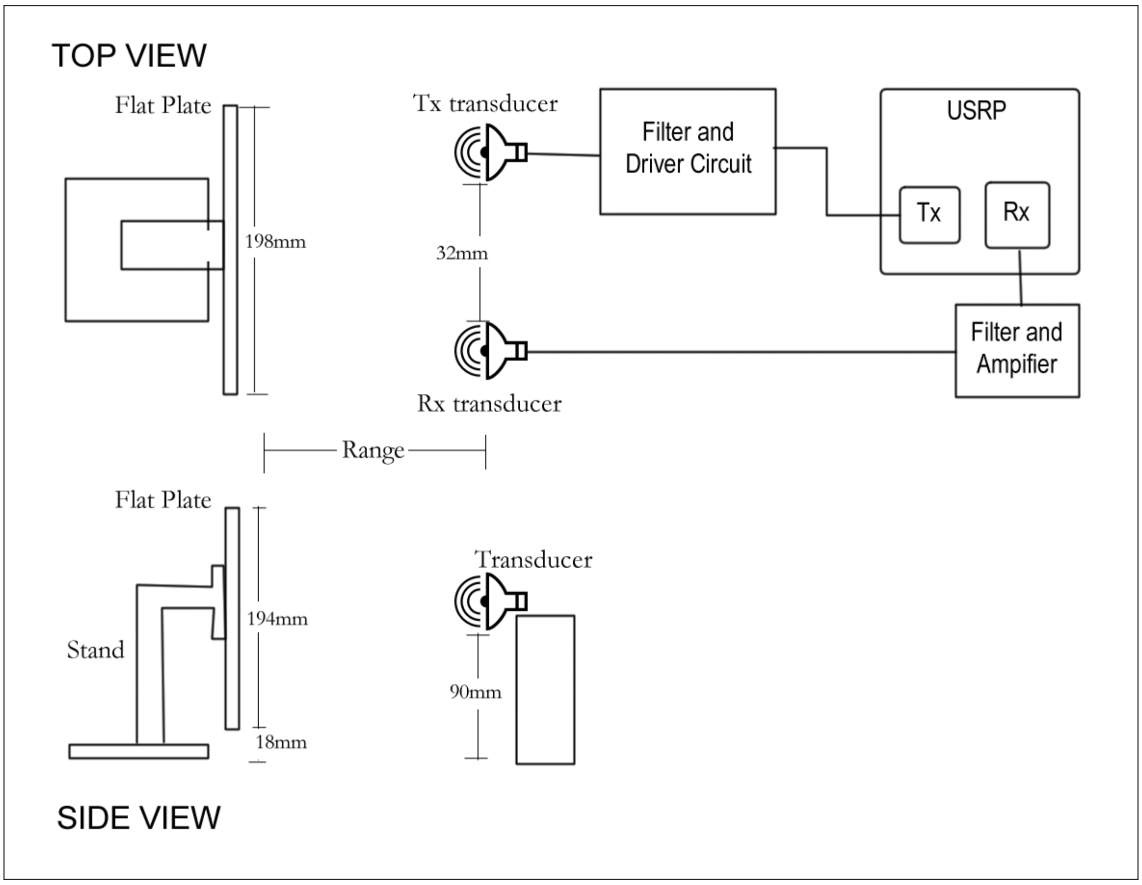
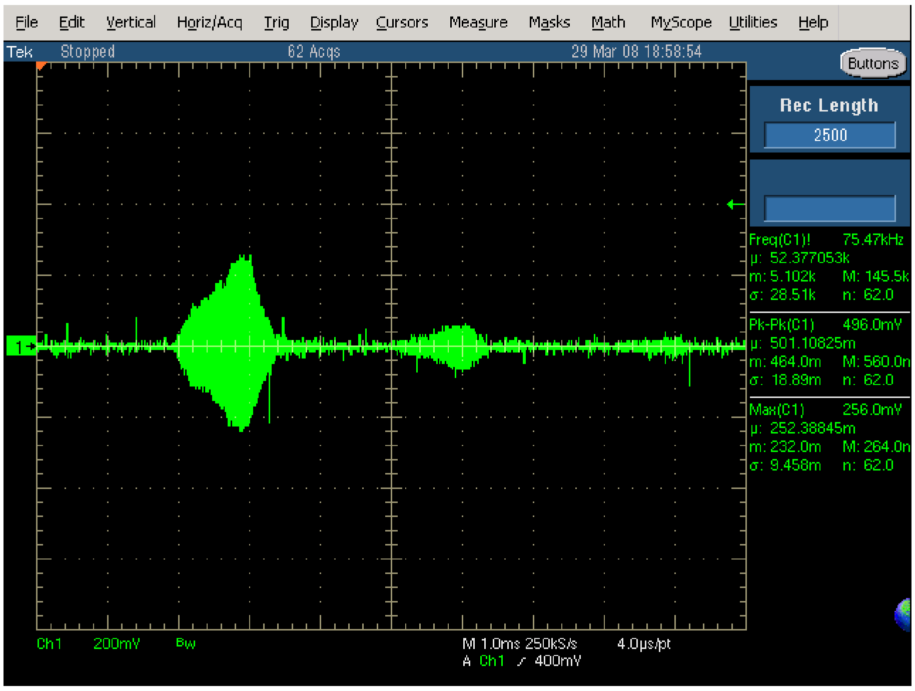


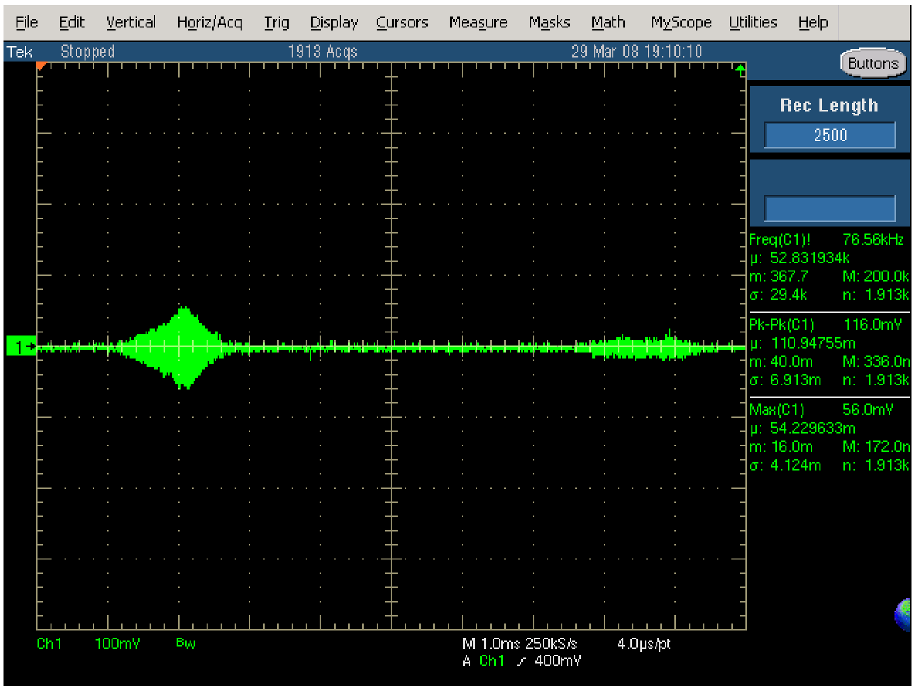
Figure 5.12: Diagram depicting the receiver circuit test setup.

Settings:	Interpolation rate = 500
	SW gain = 32e6
	PGA = 20 dB
	Δf = 39-41 kHz
	t_{pri} = 10 ms
	τ = 1 ms
	Range to target = 0.5 metre


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Buttons
Rec Length
1

```
Tek   Stopped          62 Acqs          29 Mar 08 18:58:54
Rec Length  2500
Freq(C1)!     75.47kHz
μ: 52.377053k
m: 5.102k    M: 145.5k
σ: 28.51k    n: 62.0
Pk-Pk(C1)     496.0mV
μ: 501.10825m
m: 464.0m    M: 560.0n
σ: 18.89m    n: 62.0
Max(C1)       256.0mV
μ: 252.38845m
m: 232.0m    M: 264.0n
σ: 9.458m    n: 62.0
Ch1   200mV   Bw        M 1.0ms 250kS/s    4.0μs/pt
                        A  Ch1  ⟋  400mV
```

Figure 5.13: Trace showing the analogue echo from a flat plate target 0.5 metre, as measured after the front-end.


File Edit Vertical Horiz/Acq Trig Display Cursors Measure Masks Math MyScope Utilities Help
Buttons
Rec Length
1

```
Tek   Stopped          1913 Acqs          29 Mar 08 19:10:10
Rec Length  2500
Freq(C1)!     78.56kHz
μ: 52.831934k
m: 367.7     M: 200.0k
σ: 29.4k     n: 1.913k
Pk-Pk(C1)     116.0mV
μ: 110.94755m
m: 40.0m     M: 336.0n
σ: 6.913m    n: 1.913k
Max(C1)       56.0mV
μ: 54.229633m
m: 16.0m     M: 172.0n
σ: 4.124m    n: 1.913k
Ch1   100mV   Bw        M 1.0ms 250kS/s    4.0μs/pt
                        A  Ch1  ⟋  400mV
```

Figure 5.14: Trace showing the analogue echo from a trihedral corner reflector positioned 0.5 metre from the sonar, as measured after the front-end.

Test Summary

Clear images of the acoustic reflections were successfully captured using the oscilloscope capture functionality. The traces (seen above) did not show any striking difference between the two target types in these particular tests. The flat plate target was easier to erect and position accurately. Therefore, it was used exclusively for the tests that follow.

5.4 Data Capture Tests

Purpose:

Prove the digital receiver system specifications and functionality. This will demonstrate the user control over the implemented FPGA modules and show their capability. This covers the effects of varying the receiver window or look time of the sonar. Also covered are the effects of varying the time delay between the PRI trigger pulse and the initiation of the digitisation and data transfer process. This is when receiver data is digitised and transferred to the host computer for archiving and processing.

Description:

Sample received echoes showing the effect of varying the sonar Rx window (look time) and varying the start of digitisation time relative to the PRI trigger pulse. The experiment setup is identical to that depicted in figure 5.12, with the receiver circuit and USRP Rx daughterboard being employed. The captured data is written to a binary data file. These data files are processed in Matlab which is used to generate the plots seen in the sections below. In all tests with the receiver subsystem, only a real channel was gathered from the front-end circuitry. The Basic RX boards do not contain I-Q phase shifting and this functionality was omitted from the constructed front-end circuitry for simplicity.

Settings:	Decimation rate = 250
	PGA = 20 dB
	Δf = 39-41 kHz
	t_{pri} = 10 ms
	τ = 1 ms
	Range to target = 0.5 metre

5.4.1 Receiver Window Test

This test shows the effect of varying the size of the receive window setting.

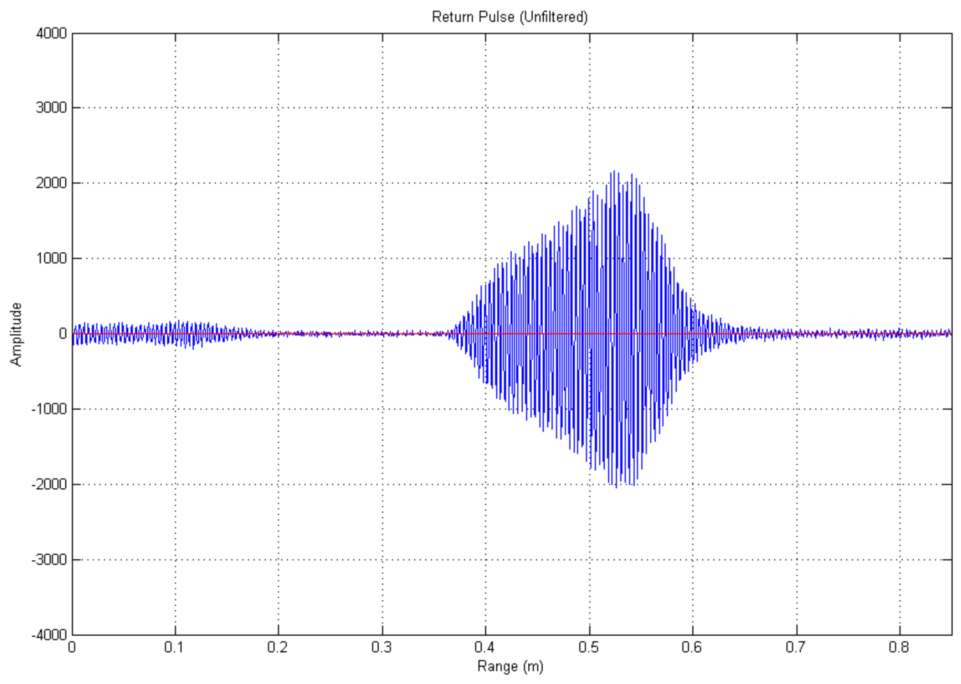


Figure 5.15: Plot showing the unfiltered response from a flat plate target. Rx window is 5 ms. The maximum viewing range is 0.85 m. Digitisation start time setting is 0 ms.

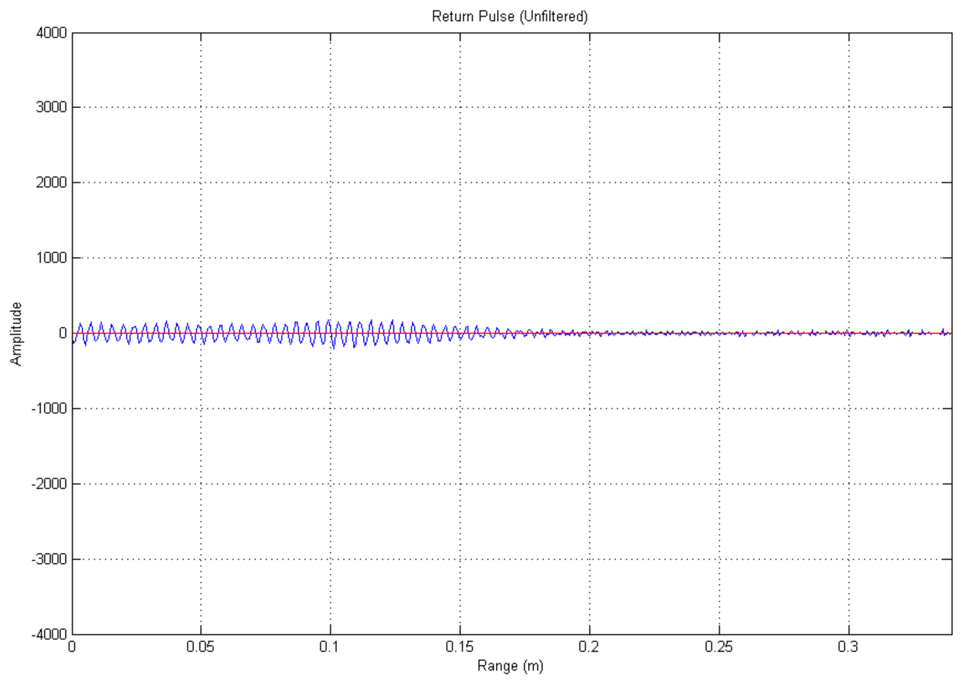


Figure 5.16: Plot showing received data with Rx window adjusted to 2 ms. The resultant viewing range is 0.34 metres. The target is no longer visible.

Test Summary

The control of the receiver window functioned as expected. By simply adjusting the look time, the maximum range that the sonar could detect targets could be predictably altered. As shown in 5.16, the target position has not been changed however, it is now beyond the sonar's viewing range.

5.4.2 Digitisation Start Time Test

The following plots depict the effect of changing the start of the data capture process relative to the PRI trigger event.

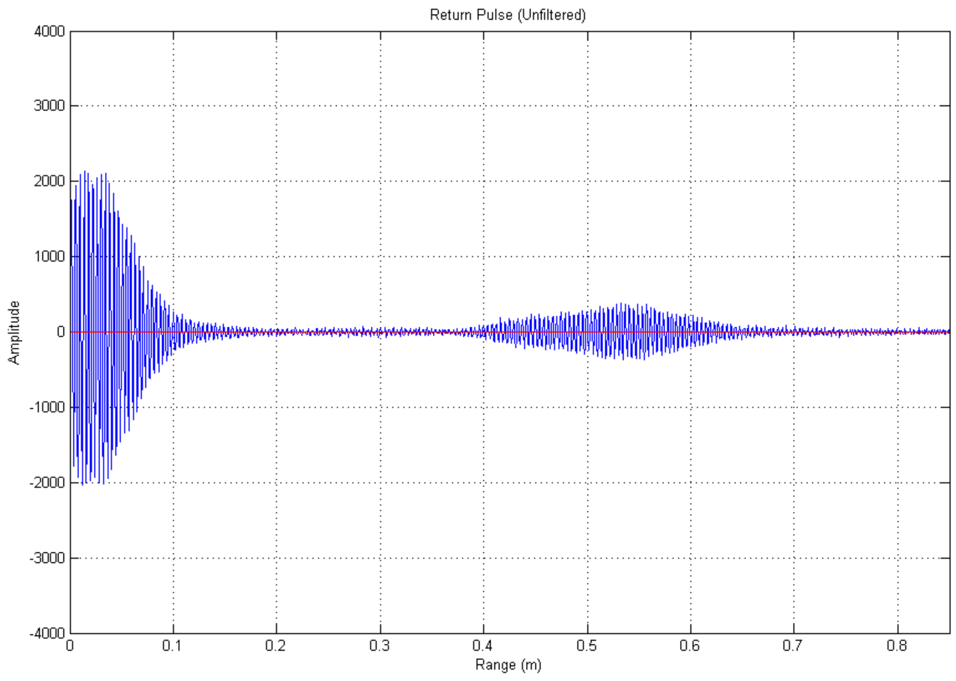


Figure 5.17: Plot showing the response from a flat plate target. Digitisation start time is 4 ms. Rx window = 5 ms. Adjusted start time discards 0.68 metres of range data. The target response is truncated.

Test Summary

As expected, a truncated target response is received as the target range falls inside the minimum detectable distance. It was noted in testing that the start time setting does not proportionally alter the Rx window as expected. An alternate implementation of this specific feature within the FPGA would correct this functionality. It was not critical to the sonar application developed in this thesis, though it would have to be corrected if radar applications are to be implemented. The data sets are relatively small and easy to control via the Rx window parameter. Additionally, the receiver circuitry is very robust and relatively low signal levels allow the system to operate within the ADC saturation values.

5.5 Data Processing Tests (Stationary Target)

Purpose:

To demonstrate the ability to capture and archive sonar data. The captured data should be usable to make accurate range calculations.

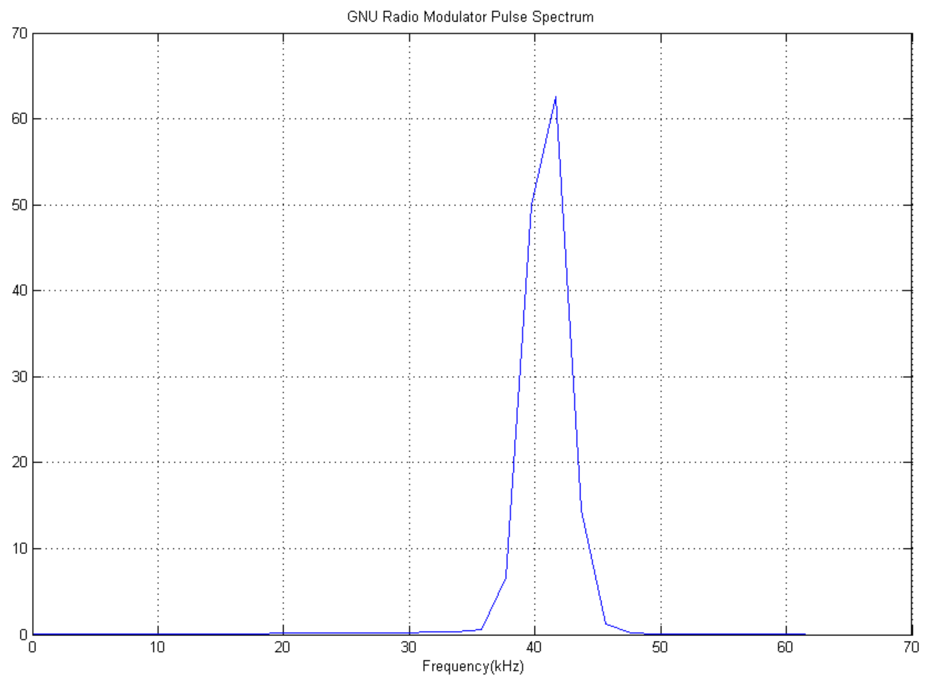
Description:

Capture reflected signal data sets for a stationary flat-plate target. Perform tests for different target ranges and utilise various system parameters to show predictable and accurate system behaviour. Captured data is processed in Matlab to produce spectrum plots. Captured data is also match filtered and output plots displayed. Due to the fact that only real channel data is gathered from the receiver circuitry, the matched filter plots depict only real signal correlations in every test. The experiment is repeated using a linear frequency chirp waveform, and an unmodulated pulse waveform. The test setup is identical to that shown in figure 5.12.

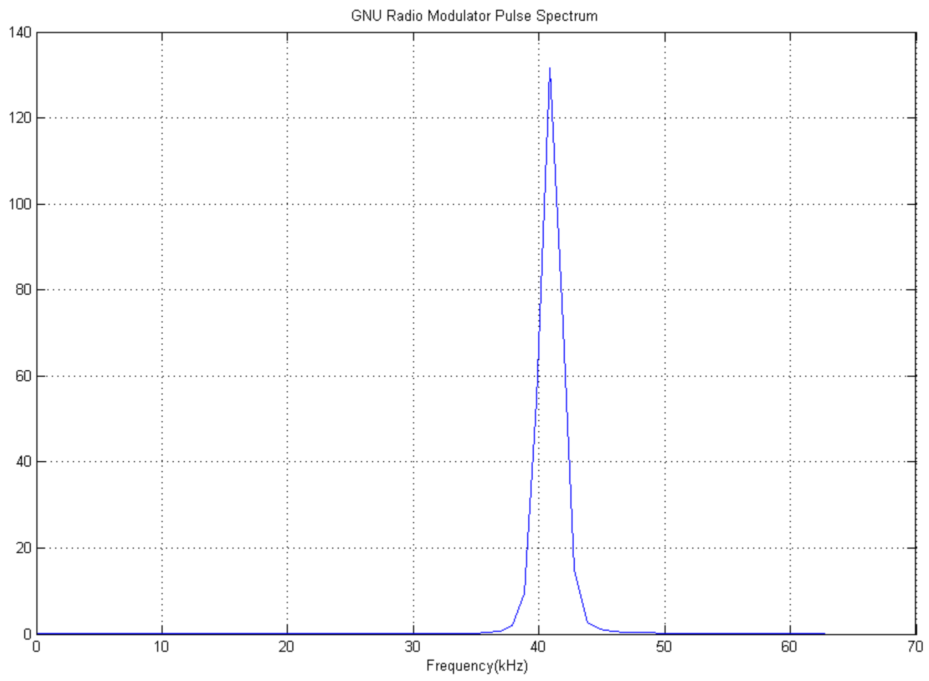
5.5.1 Linear Chirp Pulse

Settings:	Interpolation rate = 500
	Decimation rate = 250
	PGA = 20 dB
	Δf = 39-41 kHz
	t_{pri} = 10 ms

The match filters used in the following tests were created using sample sets generated from the modulator block in GNU Radio. This same code block was used to generate the chirp pulses used in the sonar application.

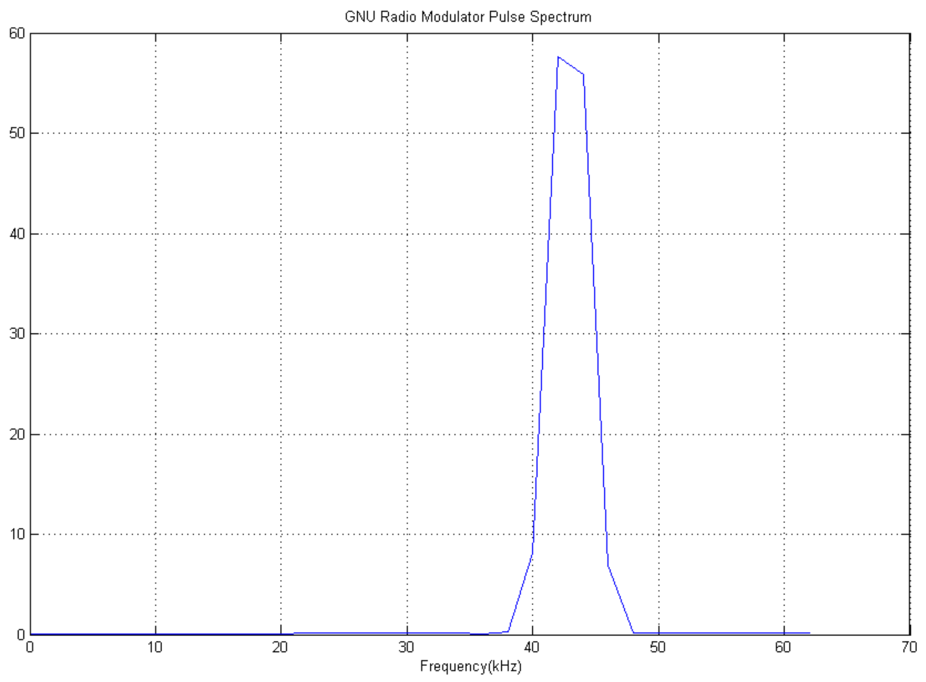


(a)

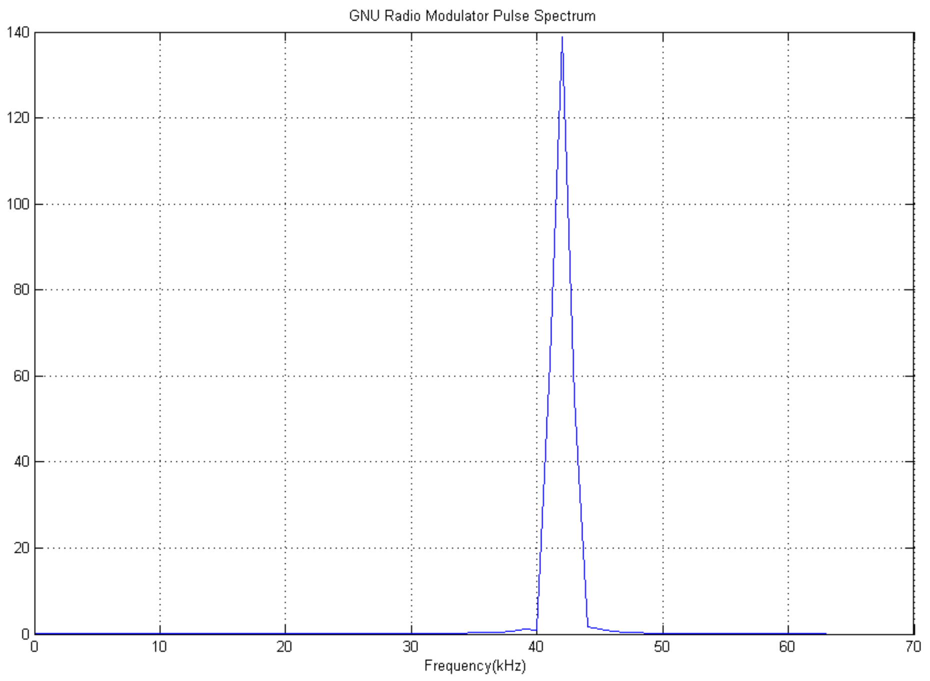


(b)

Figure 5.18: Plots showing the frequency spectrums of the sample sets used to generate the match filters for the chirp waveforms. (a) Spectrum of the set used to filter the 500 μs chirp pulses. (b) Spectrum of the set used for the 1 ms chirp pulses.



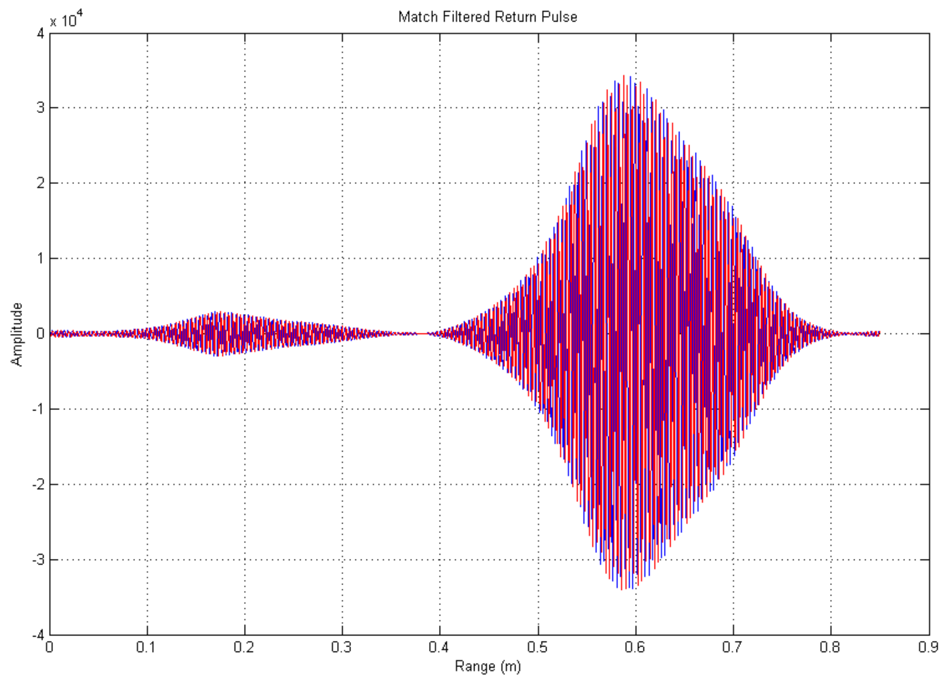
(a)



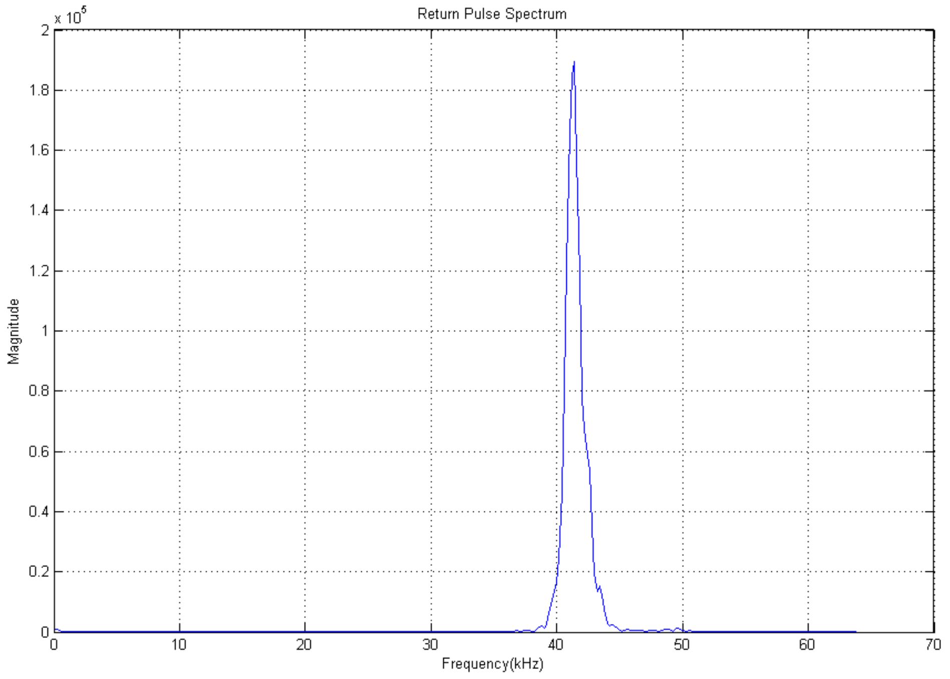
(b)

Figure 5.19: Plots showing the monotonic pulse sample set frequency spectrums. (a) Spectrum of the set used to filter the 500 μs pulses. (b) Spectrum of the set used for the 1 ms pulses.

The following plots show the responses from a flat plate target placed 0.5 metre from the sonar. The first test used a τ = 1 ms. The second uses τ = 500 μs.

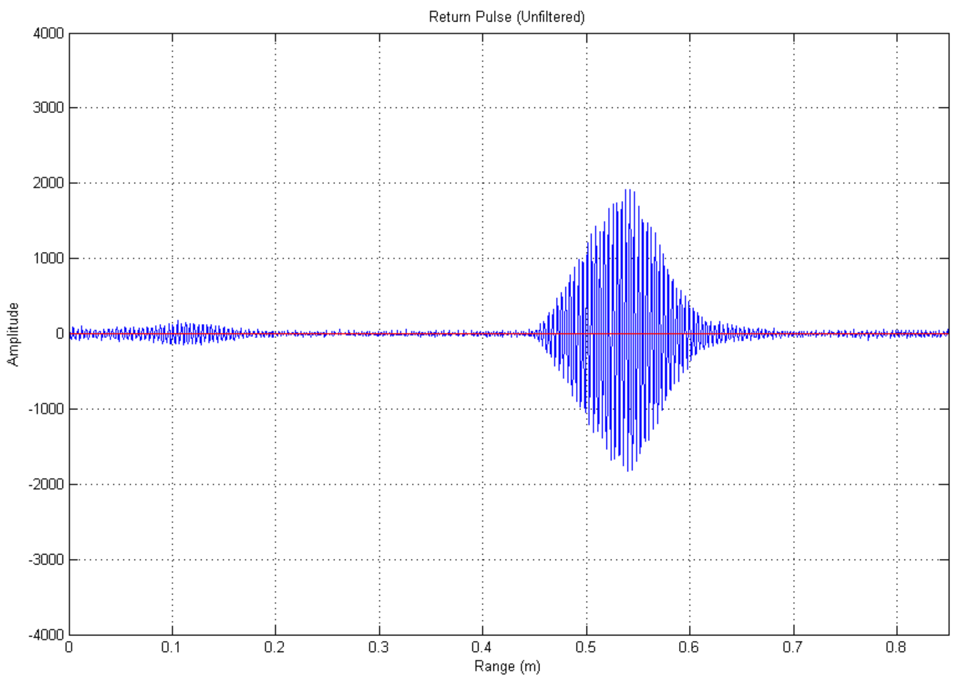


(a)

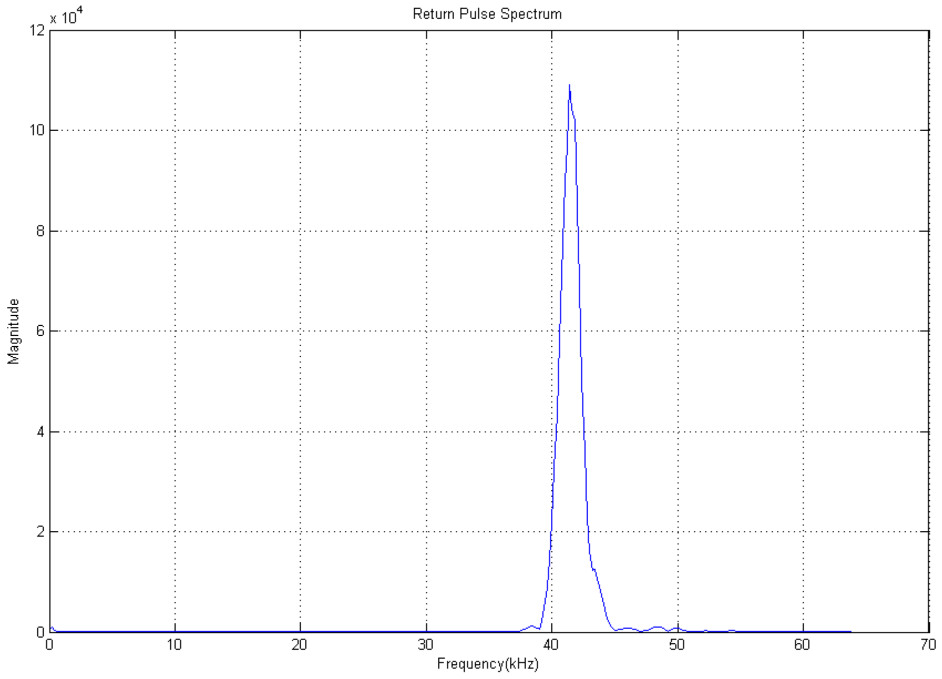


(b)

Figure 5.20: (a) Plot showing the match filtered result of the data set gathered from the test in section 5.4. (b) Plot of the response spectrum. τ = 1 ms, Rx window = 5 ms.



(a)



(b)

Figure 5.21: (a) Plot showing the unfiltered response from a flat plate target. (b) Plot of the response spectrum. τ = 500 μs, Rx window = 5 ms.

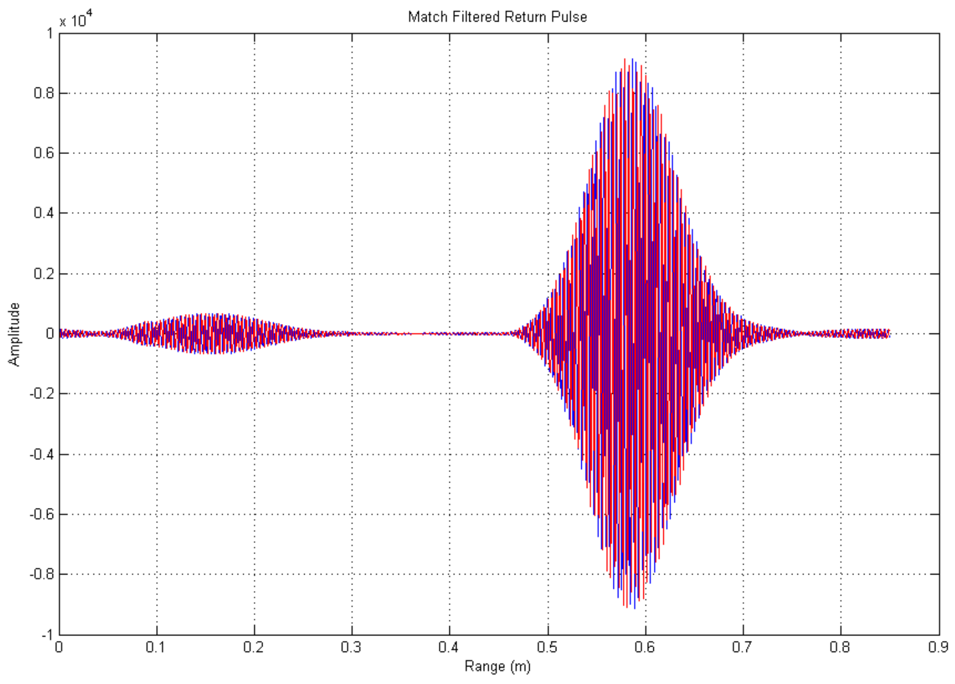
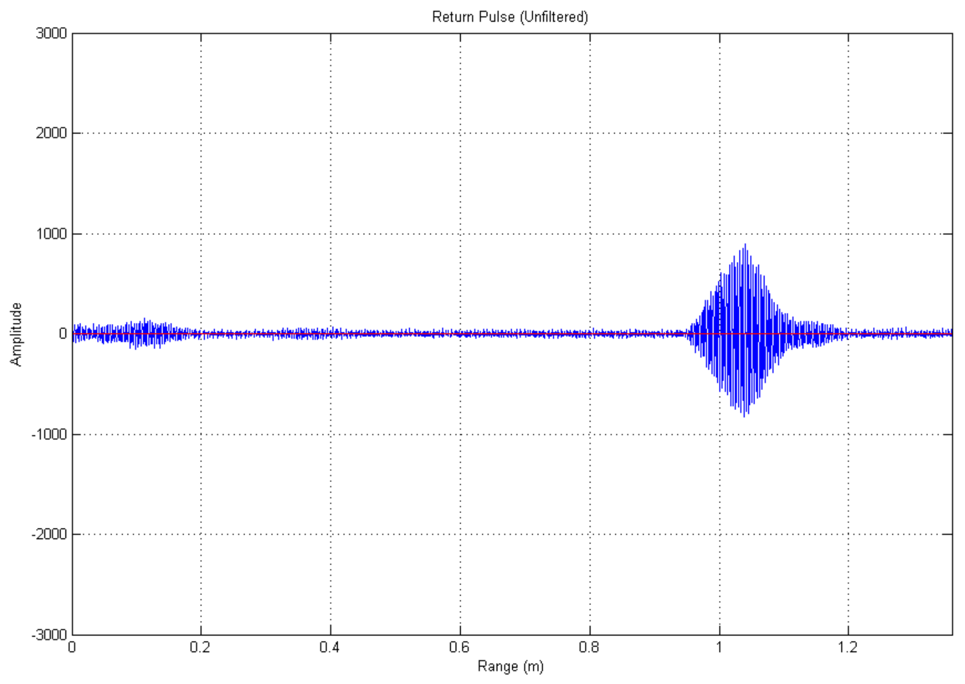
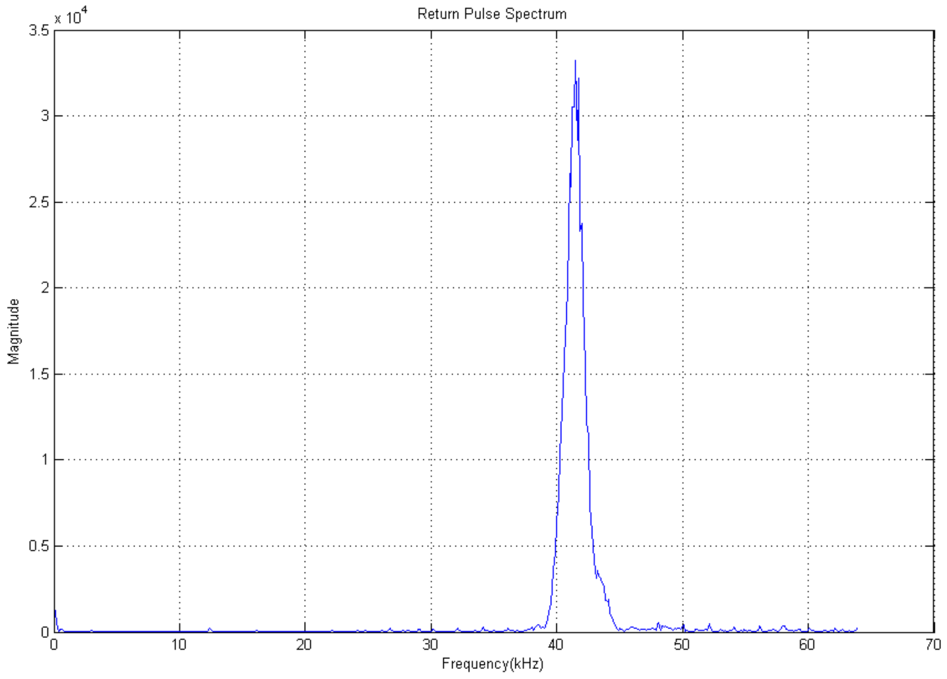


Figure 5.22: Plot showing the match filtered response from a flat plate target. $\tau = 500\ \mu s$, Rx window = 5 ms.

The following tests were performed for a flat plate target positioned 1 metre from the sonar. The Rx window value was modified for the new range, and two tests were performed with different pulse lengths.



(a)



(b)

Figure 5.23: (a) Plot showing the unfiltered response from a flat plate target. (b) Plot of the response spectrum. τ = 500 μs, Rx window = 8 ms.

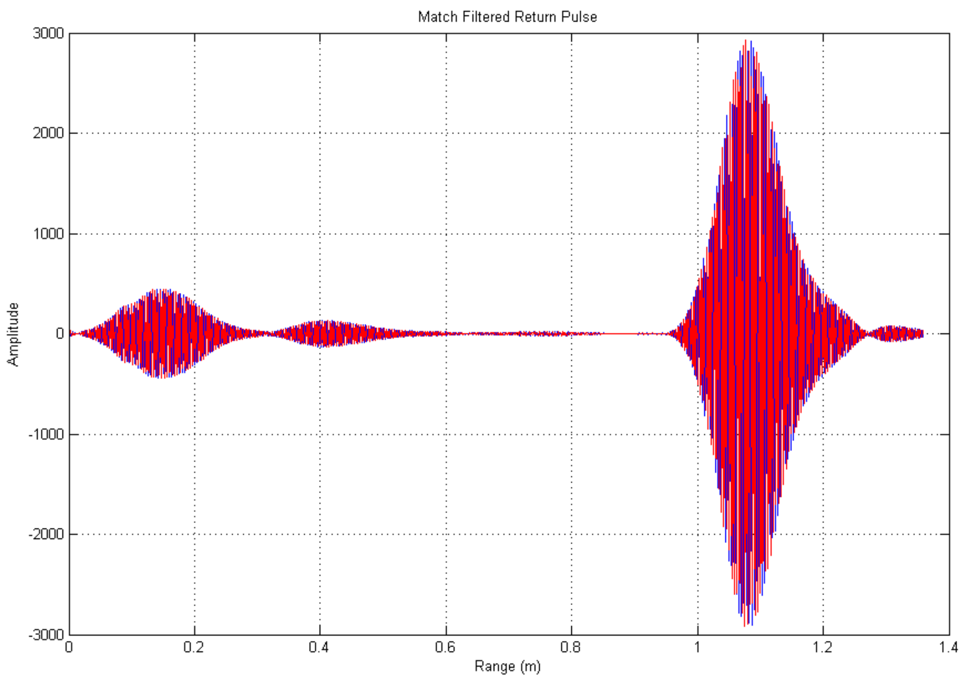
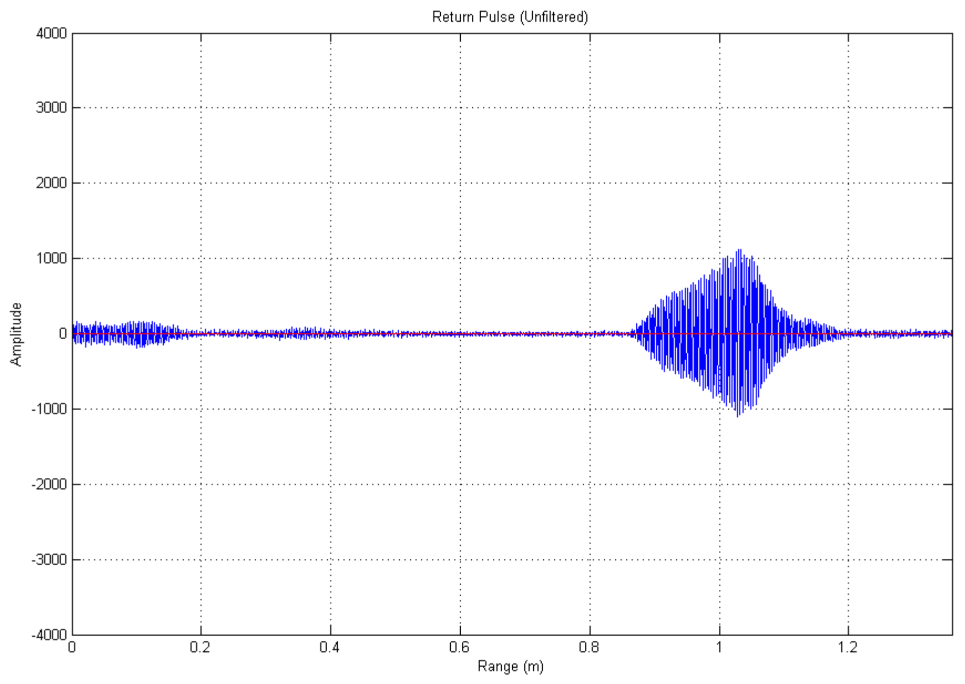
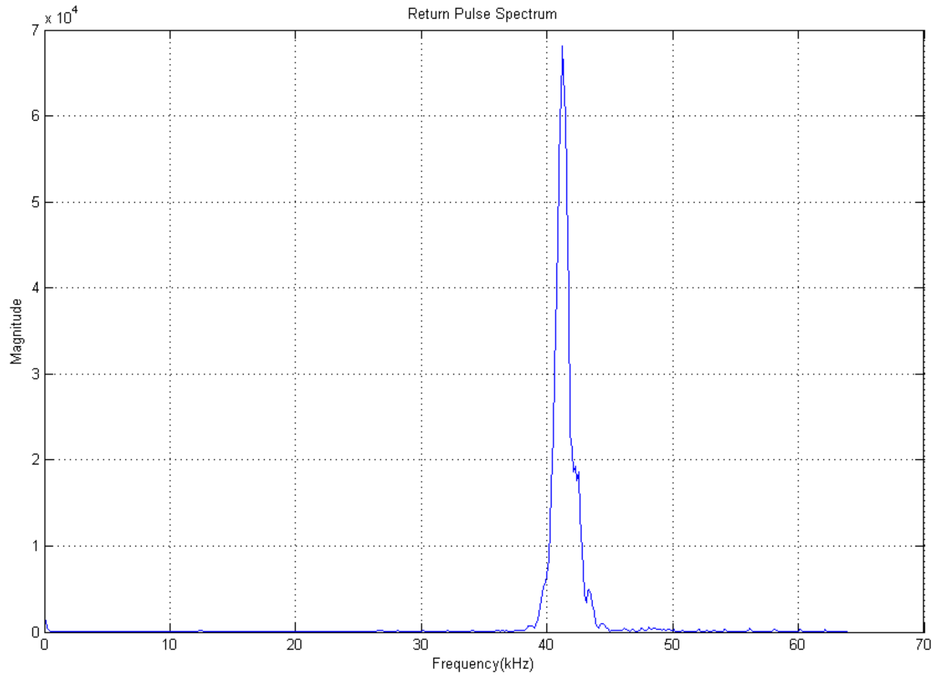


Figure 5.24: Plot showing the match filtered response from a flat plate target. $\tau = 500\ \mu s$, Rx window = 8 ms.



(a)



(b)

Figure 5.25: (a) Plot showing the response from a flat plate target. (b) Plot of the response spectrum. τ = 1 ms, Rx window = 8 ms.

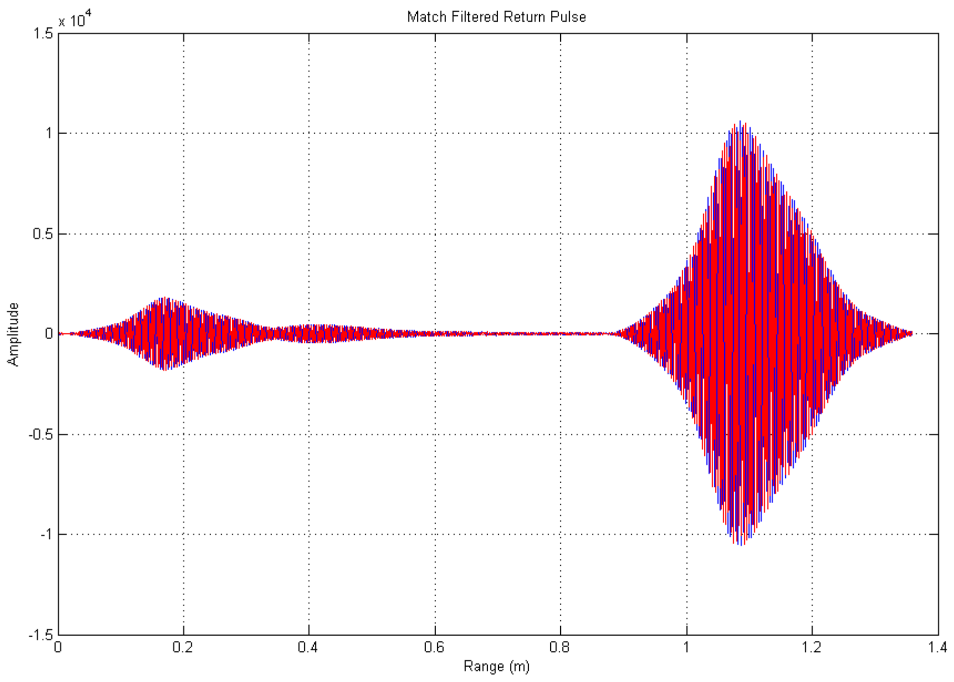
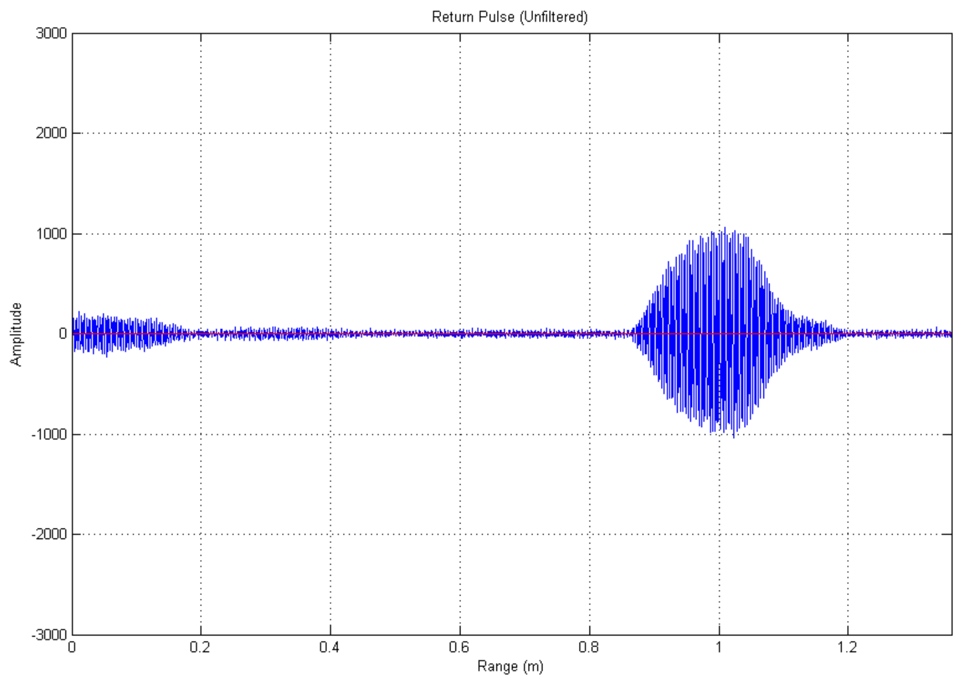


Figure 5.26: Plot showing the match filtered response from a flat plate target. τ = 1 ms, Rx window = 8 ms.

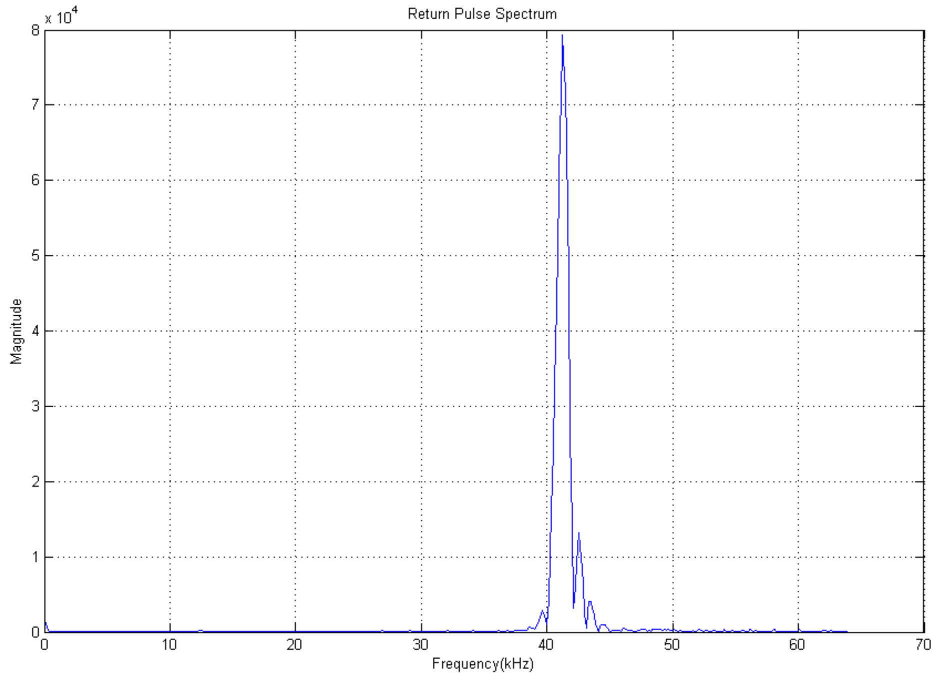
5.5.2 Unmodulated Pulse Tests

Settings:	Interpolation rate = 500
	Decimation rate = 250
	PGA = 20 dB
	f = 40 kHz
	t_{pri} = 10 ms

To show the system's ability to function with other waveform types, tests were also conducted using unmodulated pulse waveforms centred at 40 kHz and using the same hardware.



(a)



(b)

Figure 5.27: (a) Plot of the response from a flat plate target at 1 metre range. (b) Plot of the response spectrum. τ = 1 ms, Rx window = 8 ms.

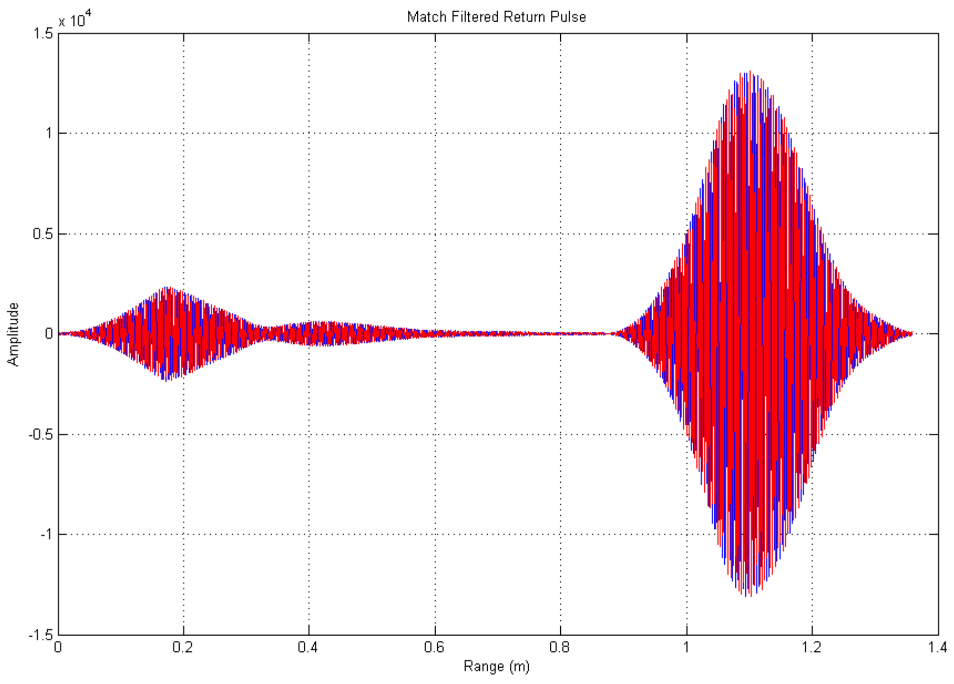
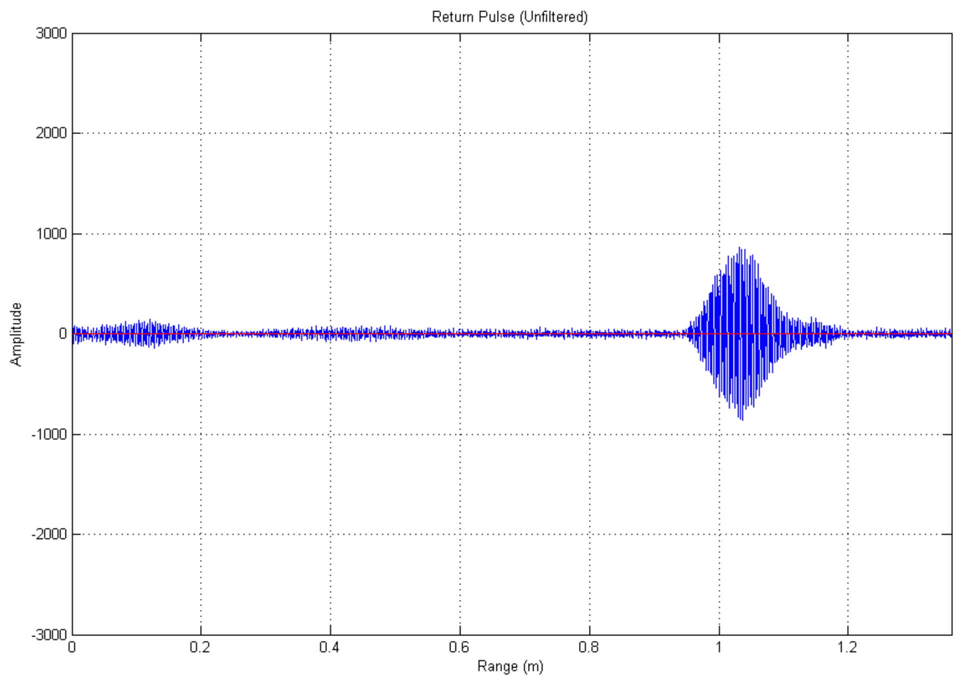
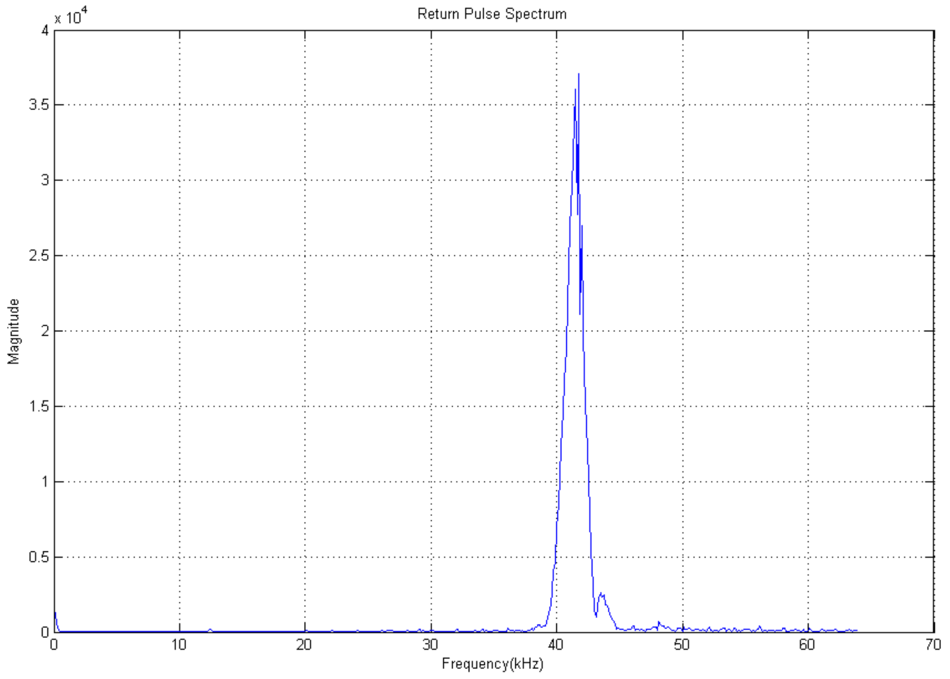


Figure 5.28: Plot showing the match filtered response from a target at 1 metre range. τ = 1 ms, Rx window = 8 ms.



(a)



(b)

Figure 5.29: (a) Plot showing the response from a flat plate target at 1 metre range. (b) Plot of the response spectrum. τ = 500 μs, Rx window = 8 ms.

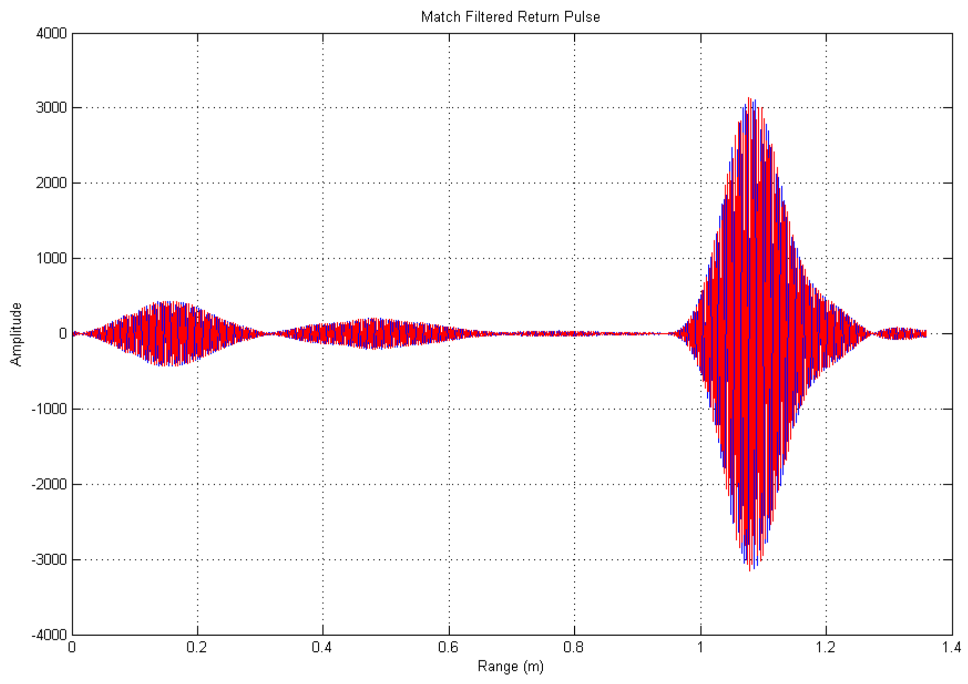


Figure 5.30: Plot showing the match filtered response from a flat plate target at 1 metre range. τ = 500 μs, Rx window = 8 ms.

5.6 Summary of Results

The output gain control, both in the software domain (GNU Radio) and the hardware domain (PGA on the USRP), performed as expected in the experiments. It was noted though that full range output from the AD9860 was not achieved from the daughterboard output connections. This was attributed to the output transformers located on the Basic Rx and Tx daughterboards, which are documented to attenuate signals below 100 kHz. This attenuation was tolerable for the testing purposes of this thesis and was unavoidable due to more suitable daughterboards being unavailable. For operation at the maximum possible gain limits of the USRP, at frequencies around 40 kHz as used in this sonar project, the LFRX and LFTX daughterboards are more suitable. These daughterboards are also available from the USRP developers and they are specifically designed to operate from DC - 30 MHz.

The PRI control functioned exactly as predicted. Users can freely choose their desired setting at a resolution of 15 ns. The pulse duration control was limited to two values (1 ms and $500\mu s$) in the final system, though continued development in the FPGA code would certainly remedy the limitation. Control of the pulse duration could also be exerted through the selected interpolation rate, though this is strongly not recommended as it causes variations in gain and distortion in the output waveform amplitude and frequency over the pulse duration.

The receiver coupling tests showed no significant direct coupling between the transmitter and receiver transducer which could be effectively eliminated with physical screening. Some signal coupling through to the receiver signal was detected, but this was attributed to circuit design and layout. For the purposes of the experiments performed in this thesis, the degree of signal coupling was tolerable.

The software-based receiver functionality developed in this thesis for the FPGA functioned mostly as expected. A fault was found in the implementation of the digitisation process initiation, which created variances in the relative time calculated with the receiver sample rate and the number of samples captured. Since this functionality was not needed for this particular sonar project, it did not affect the other tests described in this chapter. It would however be required to be functional for future radar applications.

The development of the data processing tests revealed an inaccuracy in the GNU Radio modulator block output. As seen in the plots of the modulator frequency response, there exists an offset of up 1kHz which may not be consistent. The impact can be seen in the match filtered responses, as even though the response amplitude significantly increases, the waveform remains relatively spread in time. It also meant that the transducers were not optimally driven with the most power over their resonant frequency. This would cause non-optimal power output from the transducers, which would mainly affect the maximum range the system could achieve. Due to the nature of the tests required to demonstrate the sonar concept, this non-optimal output was tolerable. This would however suggest that for expanded and more reliable control over the waveforms generated for transmission, a custom GNU Radio block would have to be developed. Such an approach would even be preferable as it allows the set of producible waveforms to be expanded.

Chapter 6

Conclusions and Recommendations

6.1 Summary

In this project, the following has been done:

- The advantages of implementing radar systems which use telecommunication components and SDR have been discussed.
- The GNU Radio framework and specifics of its functionality have been investigated.
- The operation of the USRP has been explored.
- A small scale sonar prototype for simulating a larger radar system has been developed.
- A GNU Radio program for the purpose of testing sonar and radar based GNU Radio applications has been developed.
- Various FPGA configurations have been compiled for the USRP to achieve monostatic Radar/Sonar functionality and to assist in the development of netted radar, and Passive Coherent Location (PCL) radar prototypes using the USRP.
- Front-end circuitry has been developed to support the use of ultrasonic transducers with the USRP. A short range ultrasonic sonar prototype based on the USRP has been tested.

6.2 Discussions and Conclusions

The software components of the sonar application developed in this thesis functioned as expected. The FPGA modifications also proved successful. The USRP can therefore be successfully modified from its original telecommunications purpose. By means of very little modification to the GNU Radio framework, a very basic non-real time sonar application could be developed. The USRP demonstrated that it was reliably able to capture the desired transmitted pulses.

The application did reveal certain shortcomings with the GNU Radio processing block components. In particular the modulator block was shown to be inaccurate in reproducing the requested signal. Frequency plots of the block's output showed errors that would be significant in high frequency radar systems. This indicates that development of custom processing blocks is vital if the system is to be expanded to a radar prototype. In addition, The GNU Radio framework proved to be slightly more complex to modify for pulsed, time controlled data transfer than initially expected. This problem could well be remedied however by the new GNU Radio version that is currently under development.

In looking at the theoretical abilities of the USRP and GNU Radio, they did include many features that made for interesting comparison to the UCL netted radar prototype nodes outlined in section 2.4. The available RFX range of daughterboards allow for operation in the same 2.4 GHz frequency range. While the USRP itself is unable to produce a high frequency synchronising clock, it does provide the means to override its onboard clock circuitry to enable several USRPs to be slaved to a common clock (which can be provided by one USRP in the system).

The bandwidth of 50 MHz is difficult for a USRP-based system to achieve. As shown in chapter 3 the standard USRP configuration can only achieve 8 MHz for one complex channel. The primary cause of this is the USB link, which cannot sustain a relatively high throughput. This can be eased by reducing the sample size, where selecting 8 bit samples instead of 16 bit would allow the successful real time transfer of 16 MHz of complex bandwidth. Such compromises come at the cost of dynamic range however. A more elegant solution to this problem would come in the form of the new USRP being planned for release, which replaces the USB bus with an Ethernet link.

The modifications made to the USRP in this project would allow it to achieve variable PRFs comparable to 1- 10 kHz using the implemented counter system. Pulse durations of 10 - 20 ns are also well within the capabilities of the modifications made to the USRP, where over sampling necessitated by the AD9860 chips on the USRP should produce well defined waveforms.

Below is a reiteration of the design goals listed in Chapter 2, along with a comparison with the system capabilities eventually achieved:

- Definition of known radar waveforms: This was achievable via the modulator block native to GNU Radio. Waveforms could be defined mathematically has numerical arrays corresponding to desired instantaneous frequency output. The waveforms implemented using this approach included a linear chirp waveform, and a monotonic pulse. It was noted however, that the centre frequency of the output waveforms from the modulator was not completely reliable, and showed a slight offset when the frequency spectrum was viewed. This may be related to the specifics of the algorithm used within the modulator block, which implements a phase angle accumulator to produce a rotating phase vector to generate the desired frequency.
- Accurately produce a radar waveform: The tests within the scope of this thesis only covered low frequency waveforms, though the modulation of these waveforms and the eventual outputs generally behaved as expected. It was noted also during research into the USRP's capabilities, that the modulation schemes within the USRP and particularly the AD9860 chips was more than capable to accurately produce waveforms up to intermediate frequencies.
- Provide precise PRF definition and control: this vital functionality of a pulsed radar or sonar was successfully implemented. The counter modifications were able to produce timing resolution of 15 ns.
- Control over the digital receiver subsystem including accurately defining receiver windows and controllable start of digitisation times: Both of these functions were successfully linked to the PRF, producing relative timing controls which could be compared with sample rates to resolve the precise relative times. While the receiver window behaved as expected and could be verified in the digitised data, the delay control did not function entirely as expected most likely due to a bug within the modified FPGA code. This should be resolvable however with slight changes to the implemented code.
- Provide timing control relative to the PRF: As mentioned above, this control relative to the PRF was achieved and behaved accurately in the case of the receiver window control. The sonar application tests that were performed were able to confirm that reasonably accurate range data could be resolved from the received data and using knowledge of the USRP internal sample rates as predicted.
- Provide means of reliably archiving and displaying data: The task of using GNU Radio to image received data could not be achieved. The GNU Radio synchronising thread control could not be easily accessed and new software threads could not be

readily added. This prevented correct imaging of data using the existing graphing modules, as the incoming data stream could not be properly segregated in real time to display a single return pulse window plot. This lack of external program threading ability will most likely be corrected in later releases of GNU Radio, though its absence in the version employed in this thesis project was certainly a hindrance. The system was able to very easily archive data however, and could be used to form data files which were processed in non-real time using Matlab.

- Provide means of accurately processing received data: As mentioned above, the inability to control threading within GNU Radio prevented many of its powerful processing blocks from being effectively used. The time periods between processable data arrivals, which occur in pulsed radar are the key difference compared to the streaming data telecommunications function for which GNU Radio was designed. In this sense, the USRP proved a great deal easier to modify (to perform pulsed radar applications) than GNU Radio.

6.3 Recommendations

The current version of the sonar application is non-real time, with all post-processing done in Matlab. This does not make use of the true power of the GNU Radio framework, in terms of its ability to do real time processing in software. Development of custom processing blocks and output blocks, would expand the system's functionality, and would make it a powerful teaching tool for radar subjects. As mentioned above, this would prove difficult with no easily available access to GNU Radio's synchronising thread. Newer versions of GNU Radio where application implementation is in C ++ level code should be favoured.

The above may be more viable as work in the online community which develops for GNU Radio continues, particularly the developments of the message block functionality in new versions of GNU Radio. The message block tools allow for inband signalling to take place over the USB link. This has the potential for not only making data capture and display easier, but also for adding more advanced radar functionality. Particularly capabilities like dynamic PRFs, and the ability to change transmit waveform characteristics dynamically, would make the simulation of much more advanced radar systems possible.

Newer versions of the USRP itself are also under development. These more powerful variants provide more for radar and sonar developers, particularly in terms of:

- Overall functionality.
- More powerful FPGAs capable of performing faster calculations.

- Larger FPGAs which can do more processing.
- Faster USRP to desktop links which can provide higher signal bandwidths.

An investigation into various means of accelerating GNU Radio's operation may also be very worthwhile for the development of more powerful radar systems using this device later on. By running various processing algorithms such as signal correlation in add-on hardware such as a graphics card's Graphical Processing Unit, a system running a GNU Radio based radar application may be able to achieve much more processing capability. Handling more rigorous processing algorithms without a loss in performance would become possible. By using more intensive post-processing techniques, it becomes possible to gain performance in some cases, even if limited hardware such as the USRP is being used.

Development of custom daughterboards for the USRP is also an interesting possibility. This gives the chance to achieve more tailored or better performing systems. It also allows for adding more desirable functions for the purpose of radar or sonar systems.

Development of USRP based prototype radar systems is the larger goal of which this project only forms a part. Development of PCL and netted radar systems is the next step from this project, as it would allow for important tests to be conducted into these radar research fields.

Appendix A

GNU Radio Script File

This file is setup to use the custom FPGA builds. By placing it in the rev4 folder in /usr/local/share/usrp, rev 4.0 usrps should be able to load it. Currently it produces a chirp which sweeps from 39 to 41 kHz. To change this, modify the vec1 vector. This vector determines what signal the gr.frquency modulator block produces. This block will take in the vector (which defines the real part) and output a complex signal. Currently, the FPGA build will only accept sample set sizes of 512 and 256.

```
#!/usr/bin/env python
from gnuradio import gr, gru
from gnuradio.wxgui import stdgui, fftsink, form, slider, scopesink
from gnuradio import eng_notation
from gnuradio import usrp
from gnuradio.eng_option import eng_option
from optparse import OptionParser
import wx
import math
import sys
import Numeric
import usrp_dbid
from usrp_fpga_regs import *
import experi_plot2
//
class ptt_graph(stdgui.gui_flow_graph):
    def __init__(self, frame, panel, vbox, argv):
        stdgui.gui_flow_graph.__init__ (self, frame, panel,
                                        vbox, argv)
        self.frame = frame
        #
        print "GUI building"
        self._build_gui(frame, panel, vbox, argv, options.no_gui)
    def set_transmit(self, enabled):
        self.txpath.set_enable(enabled)
        self.rxpath.set_enable(not(enabled))
    def _build_gui(self, frame, panel, vbox, argv, no_gui):
        self.panel = panel

        vbox.Add(wx.Button(panel, -1, 'Button1'), 1, wx.ALL | wx.ALIGN_CENTER |
                wx.EXPAND, 5 )
        vbox.Add(wx.Button(panel, -1, 'Button2'), 1, wx.ALL | wx.ALIGN_CENTER |
                 wx.EXPAND, 5 )
        vbox.Add(wx.Button(panel, -1, 'Button3'), 1, wx.ALL | wx.ALIGN_CENTER |
                 wx.EXPAND, 5 )

        panel.SetFocus()

class transmit_path(gr.hier_block):
    def __init__(self, fg):
        interp = 500        #interp rate to produce 1 ms waveform
        nchan = 1           #number of complex channels

      #duc1 set to zero because the waveform is being created at baseband.
        #Change this value to make use of the mixer in the AD9860.
        duc1 = 0
        fs = 256e3                              #2nd sample rate usb -> dac
        max_dev = 32e3                          #1st sample rate
        self.gain = 32e3
        self.pri_time = 0.01                    #pri in seconds
        self.pri_ticks = int(self.pri_time*64e6)
        k = 2 * math.pi * max_dev / fs
#
        #monochromatic pulse 40kHz [1.28]*256
        vec1 = Numeric.arange (1.248, 1.312, 0.00025)
        #vector source block takes in linear array to make chirp.
        vsource = gr.vector_source_f(vec1, False)

        thr1 = gr.throttle(gr.sizeof_float, fs)
        #fmmod generates chirp samples
        fmmod = gr.frequency_modulator_fc (k)
        #self.amp adds software gain
        self.amp = gr.multiply_const_cc(self.gain)
        #creates sink and specifies custom FPGA build to use.
        usrp_tx = usrp.sink_c (0, interp, nchan, fpga_filename =
                               'usrp_std_27-11-07(2).rbf' )
        #setup to use BASIC_TX_B daughterboard
        tx_subdev_spec = usrp.pick_tx_subdevice(usrp_tx)
        m = usrp.determine_tx_mux_value(usrp_tx, tx_subdev_spec)
        self.subdev = usrp.selected_subdev(usrp_tx, tx_subdev_spec)
        self.subdev.set_gain(0)                             # set max Tx gain
        usrp_tx.set_mux(m)
        #sets IF for mixer on AD9860.
        usrp_tx.set_tx_freq (1, duc1)
        #sets rx window size in terms of rx samples
        usrp_tx._write_fpga_reg(usrp.FR_USER_12, 2048)

        #sets maximum samples to send. related to padding.
        usrp_tx._write_fpga_reg(usrp.FR_USER_13, 256)

        #sets prf value. 20 bit was 16ms.
        usrp_tx._write_fpga_reg(usrp.FR_USER_14, self.pri_ticks)
        #sets delay value in rx samples to discard after the prf has occured.
        usrp_tx._write_fpga_reg(usrp.FR_USER_15, 0)

        #setup and enable debug output(disabled).
        #usrp_tx._write_oe(1, 0xffff, 0xffff)
        #usrp_tx._write_fpga_reg(FR_DEBUG_EN, bmFR_DEBUG_EN_TX_B)

        print "transmitter connecting now"
        fg.connect (vsource, thr1, fmmod, self.amp, usrp_tx)
        gr.hier_block.__init__(self, fg, None, None)
    def set_enable(self, enable):
      # set H/W Tx enable
        self.subdev.set_enable(enable)
        if enable:
            self.amp.set_k (self.gain)
        else:
            self.amp.set_k (self.gain)
class receive_path(gr.hier_block):
    def __init__(self, fg):
        # build the graph
        self.u = usrp.source_c(decim_rate=250, nchan=1 , fpga_filename =
                               'usrp_std_27-11-07(2).rbf' )
        filename = "output.bin"
        nsamples = 128000
        self.RX_fs = 256e3
        rx_subdev_spec = usrp.pick_rx_subdevice(self.u)
        self.u.set_mux(usrp.determine_rx_mux_value
                       (self.u, rx_subdev_spec))

        # determine the daughterboard subdevice we're using
        self.subdev = usrp.selected_subdev(self.u, rx_subdev_spec)
        input_rate = self.u.adc_freq() / self.u.decim_rate()

        # gain / mute block
        self.io_gain = gr.multiply_const_cc(1)
        self.head = gr.head(gr.sizeof_gr_complex, int(nsamples))

        #filesink
        self.dst = gr.file_sink(gr.sizeof_gr_complex, filename)

        #dead = gr.null_sink(gr.sizeof_gr_complex)
        fg.connect(self.u, self.io_gain, self.head, self.dst)
        gr.hier_block.__init__(self, fg, None, None)

        g = self.subdev.gain_range()
        gain = g[0]
        self.subdev.set_gain(gain)
        freq = 0
        r = self.u.tune(0, self.subdev, freq)
        if r:
            print "receiver ready..."
        else:
            sys.stderr.write('Failed to set frequency\n')
            raise SystemExit, 1
    def set_enable(self, enable):
        self.enabled = enable
def main():
    app = stdgui.stdapp(ptt_graph, "SONAR Application")
app.MainLoop()
if __name__ == '__main__':
    main()
```

Appendix B

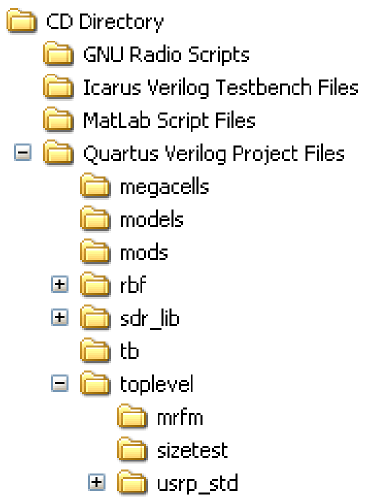
RAM Control Module Verilog Source Code

```
‘include "../../../firmware/include/fpga_regs_standard.v"
module ram_dual_alt
  ( input clock, input reset, input enable,
    input wire [3:0] channels,
    input [15:0] fifo_samples,
    input txstrobe, input sample_strobe, input rx_strobe,
    input wire tx_empty,
    input wire [6:0] serial_addr, input wire [31:0] serial_data,
    input wire serial_strobe,
    output reg [15:0] tx_i_0, output reg [15:0] tx_q_0,
    output reg [15:0] tx_i_1, output reg [15:0] tx_q_1,
    output pulsing_enable, output alt_rx_en,
    output transmitting, output [15:0] debugbus
    );
    parameter ADDR_WIDTH = 12;
    parameter MAX_SAMPLES = 2048;
    parameter PULSE_SAMPLES = 2048;
    parameter WIDTH = 32;
    parameter PULSE_DELTA = 4294967295;

    wire [ADDR_WIDTH-1:0] wr_addr;
    wire [ADDR_WIDTH-1:0] rd_addr;
    wire [ADDR_WIDTH-1:0] maximum_samples;  //set by user_reg_13
    wire [WIDTH-1:0] prf_count;
    wire [WIDTH-1:0] rx_count;
    wire [WIDTH-1:0] rx_window_size;        //set by user_reg_12
    wire rd_enable, wr_enable;
    wire RAM_Loaded, TX_Complete;
    wire read_reset, write_reset;
    wire start, dev_reset, prf_reset, delay_reset;
    wire [31:0] set_prf;                    //set by user_reg_14
    wire [31:0] set_delay;                  //set by user_reg_15
    wire [31:0] delay_count;

    reg [15:0] ramdata;
    reg [15:0] dud;
    reg [3:0]  load_next;
    reg [ADDR_WIDTH-1:0] addr;
```

```
//Assignments
    assign    rd_enable = ((load_next != channels) & tx_empty & RAM_Loaded);
    assign    wr_enable = (enable & ~tx_empty);
    assign    write_reset = (reset)?(1'b1):(wr_addr === 9'bx)?(tx_empty):1'b0;
    assign    read_reset = (reset)?(1'b1):(rd_addr === 9'bx)?1'b1:
                           (start === 1'bx)?1'b1:start;
    assign    RAM_Loaded = (wr_addr == maximum_samples);
    assign    TX_Complete = (rd_addr == maximum_samples);
    assign    dev_reset = ~RAM_Loaded;
    assign    start = (prf_count == 0);
    assign    pulsing_enable = RAM_Loaded;
    assign    prf_reset = (prf_count == set_prf);
    assign    delay_reset = (delay_count >= set_delay);
    assign    alt_rx_en = ((rx_count <= rx_window_size)
                            & RAM_Loaded & delay_reset);
    assign    transmitting = ~TX_Complete;
//
 setting_reg #(`FR_USER_12) user_reg_12(
   .clock(clock),.reset(reset),.strobe(serial_strobe),
   .addr(serial_addr),.in(serial_data),.out(rx_window_size));

  setting_reg #(`FR_USER_13) user_reg_13(
   .clock(clock),.reset(reset),.strobe(serial_strobe),
   .addr(serial_addr),.in(serial_data),.out(maximum_samples));

  setting_reg #(`FR_USER_14) user_reg_14(
   .clock(clock),.reset(reset),.strobe(serial_strobe),
   .addr(serial_addr),.in(serial_data),.out(set_prf));

  setting_reg #(`FR_USER_15) user_reg_15(
   .clock(clock),.reset(reset),.strobe(serial_strobe),
   .addr(serial_addr),.in(serial_data),.out(set_delay));
//
    //prf counter set by user_reg_14
  countert #(WIDTH, PULSE_DELTA, 0) prf_counter
    ( .clk(clock),
      .enable(RAM_Loaded),
      .reset(reset|prf_reset),
      .out(prf_count)
    );
   //delay counter set by user_reg_15
  countert #(WIDTH, PULSE_DELTA) delay_counter
    ( .clk(clock),
      .enable(RAM_Loaded & rx_strobe),
      .reset(reset|start),
      .out(delay_count)
    );
    //rx decim is 250. rate is 256e3/sec -> 256/msec.
    //Count 2560 samples for 10 msec
    //RX Chain enable
  countert #(WIDTH, PULSE_DELTA) receive_en
    ( .clk(clock),
      .enable(RAM_Loaded & rx_strobe & delay_reset),
      .reset(~TX_Complete),
      .out(rx_count)
    );
    //advances write address pointer
  countert #(ADDR_WIDTH, MAX_SAMPLES) wraddr_counter
    ( .clk(clock),
      .enable(wr_enable),
      .reset(write_reset),
      .out(wr_addr)
    );
    //advances read address pointer
  countert #(ADDR_WIDTH, PULSE_SAMPLES) rdaddr_counter
    ( .clk(clock),
      .enable(rd_enable),
      .reset(read_reset),
      .out(rd_addr)
    );
    //output to the tx chains
    always @(posedge clock)
       if(reset|dev_reset)
          begin
             {tx_i_0,tx_q_0,tx_i_1,tx_q_1}
               <= #1 64'h0;
             load_next <= #1 4'd0;
          end
       else if(TX_Complete)
          begin
              {tx_i_0,tx_q_0,tx_i_1,tx_q_1}
               <= #1 64'h0;
          end
       else
       if(rd_enable)
          begin
             load_next <= #1 load_next + 4'd1;
             case(load_next)
                4'd0 : tx_i_0 <= #1 ramdata;
                4'd1 : tx_q_0 <= #1 ramdata;
                4'd2 : tx_i_1 <= #1 ramdata;
                4'd3 : tx_q_1 <= #1 ramdata;
                default : dud <= #1 ramdata;
             endcase // case(load_next)
          end
       else if(txstrobe & (load_next == channels))
          begin
             load_next <= #1 4'd0;
          end

    //ram module
  signal_ram signal_ram1
     ( .data(fifo_samples),
           .wren(wr_enable),  //???
           .wraddress(wr_addr),
           .rdaddress(rd_addr),
           .rden(rd_enable),
           .clock(clock),
           .q(ramdata)
       );
   // Debugging Aids
   //assign debugbus bits to signals you wish to monitor
   //
endmodule
```

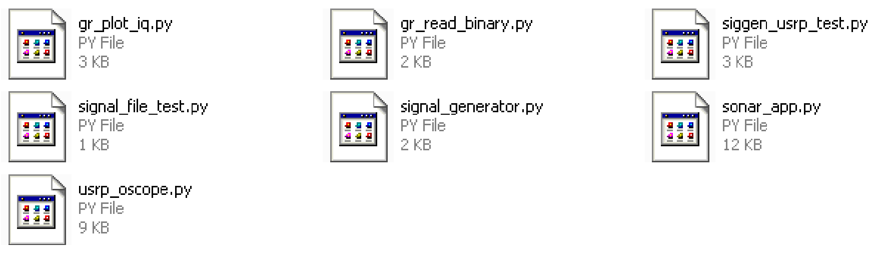
Appendix C

Project Files CD Directory Structure



```
CD Directory
    GNU Radio Scripts
    Icarus Verilog Testbench Files
    MatLab Script Files
  ⊟ Quartus Verilog Project Files
        megacells
        models
        mods
      ⊞ rbf
      ⊞ sdr_lib
        tb
      ⊟ toplevel
            mrfm
            sizetest
          ⊞ usrp_std
```

Figure C.1: Project files directory structure on the accompanying compact disk.



gr_plot_iq.py PY File 3 KB	gr_read_binary.py PY File 2 KB	siggen_usrp_test.py PY File 3 KB
signal_file_test.py PY File 1 KB	signal_generator.py PY File 2 KB	sonar_app.py PY File 12 KB
usrp_oscope.py PY File 9 KB		

Figure C.2: GNU Radio Scripts Folder - contains the sonar application script file (sonar_app.py) in addition to other test scripts for generating signals using the GNU Radio and the USRP, and testing the various blocks used in the project.

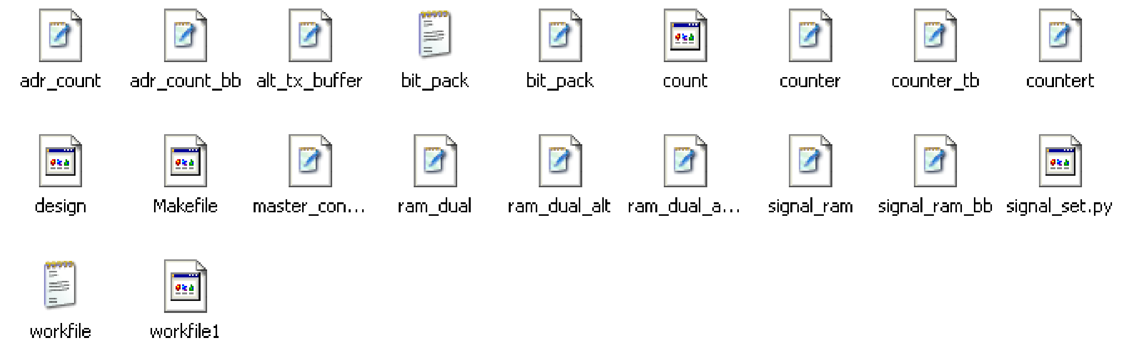


Figure C.3: Icarus Verilog Testbench Files - contains the Verilog test files used for testing the logical functionality of the modified USRP FPGA files before implementing them in the FPGA.

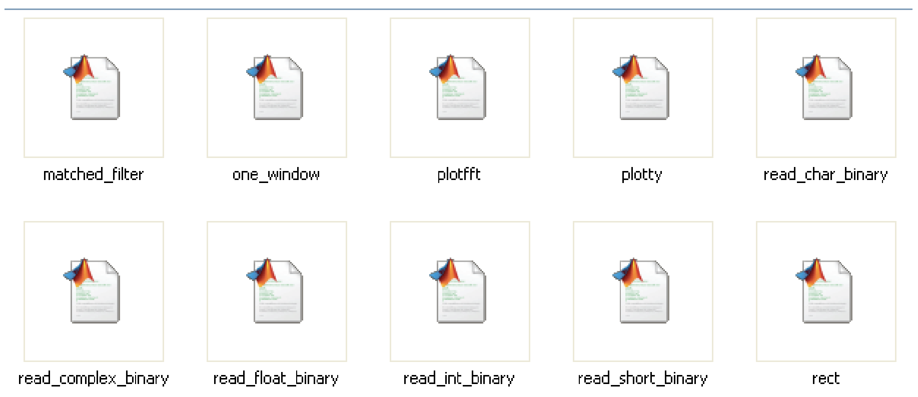


Figure C.4: Matlab Script Files - contains the files used to process data files generated by the sonar application. The “read_binary...” function is first applied on the data files, and then the “One_window” script file is used to calculate and display the desired parameters.

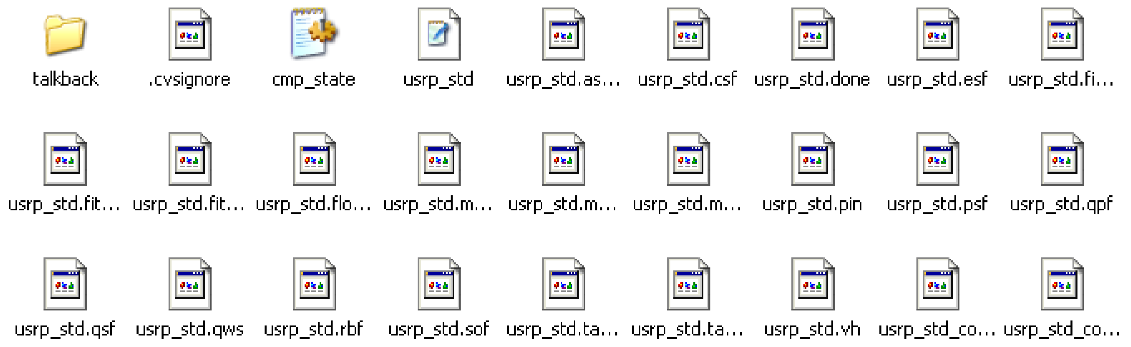


Figure C.5: Quartus Verilog Project Files - contains the main project and Verilog files required by Quartus to compile a new FPGA bitstream image. In addition to this folder, the register reference files are required from the firmware folder (which can be obtained with a download of the latest GNU Radio compressed package file) to compile a new image. Depicted are the contents of the ../top_lvl/usrp_std/ directory.

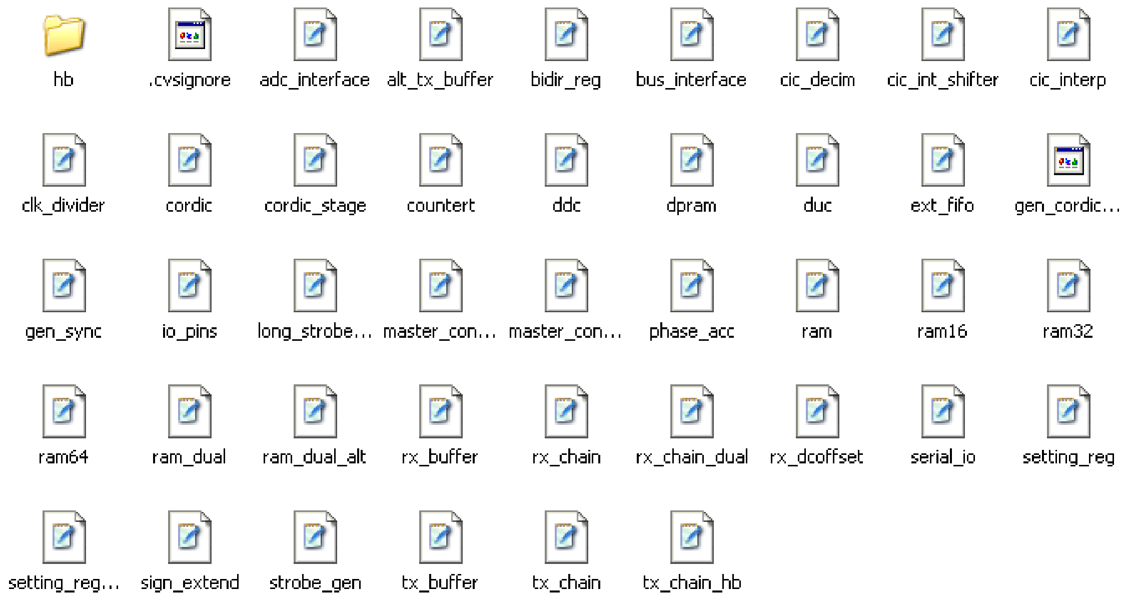


Figure C.6: Depiction of the contents of the sdr_lib folder.

The top level Verilog file which unifies all the subsystems implemented in the FPGA is called usrp_std.v, and is located in ../toplevel/usrp_std/ . The bulk of the Verilog code files that were modified and created to implement the pulsed radar functionality are located in ../sdr_lib/ . Appendix B is an extract from the ram_dual_alt.v file, located in the sdr_lib folder.

Bibliography

[1] D. McCarthy, “Software-Defined Radio: Integration for Innovation,” *RF Design*, vol. Software-Defined Radio, pp. 44–50, September 2005.

[2] R. N. Mutagi, “Understanding the Sampling Process,” *RF Design*, vol. Mixed Signal, pp. 38–48, September 2004.

[3] S. Kingsley and S. Quegan, *Understanding Radar Systems*. SciTech Publishing, Inc., 1999.

[4] Y. Teng, H. Griffiths, C. Baker, and K. Woodbridge, “Netted radar sensitivity and ambiguity,” *Radar Sonar Navig.*, vol. 1, pp. 479–486, 2005.

[5] F. G. Stremler, *Introduction to Communication Systems*. Addison-Wesley Publishing Company, Inc., 1992.

[6] S. Grossman, “Software Defined Radio poses major challenges for hardware and software developers,” *RF Design*, vol. Defense Electronics, pp. 10–15, June 2005.

[7] A. Rudra, “FPGA-based Applications for Software Radio,” *RF Design*, vol. Signal Processing, pp. 24–35, May 2004.

[8] T. Derham, S. Doughty, K. Woodbridge, and C. Baker, “Realisation and evaluation of a low cost netted radar system,” *Radar, 2006. CIE ’06. International Conference on*, pp. 1–4, 2006.

[9] T. Derham, K. Woodbridge, H. Griffiths, and C. J. Baker, “The design and development of an experimental netted radar system,” *Radar Conference, 2003. Proceedings of the International*, pp. 293–298, 2003.

[10] C. Baker, B. Bates, L. Williams, M. Inggs, L. Tao, Y. Hong, J. Zhu, W. Rossum, A. Huizing, R. Lane, and S. Hayward, “Poster Session 1c: Multistatic Radar,” *RADAR 2007*, pp. 6–7, 2007.

[11] E. Blossom, “An Introduction to Software Radio,” USENIX/Boston, June 2006.

[12] G. N. U. Radio, "USRP - GNU Radio - Trac," 2008. Available: `http://gnuradio.org/trac/wiki/USRP` [Accessed 25 July 2008].

[13] D. Shen, "Tutorial 4: The USRP Board." June 2005.

[14] Altera Corporation, *Cyclone Device Handbook*, January 2007.

[15] Analog Devices, *Mixed-Signal Front-End Processor for Broadband Communications- AD9860*, 2002. Rev. 0.

[16] M. Ettus, "Ettus Research LLC," 2008. Available: `http://www.ettus.com` [Accessed 25 July 2008].

[17] M. P. Donadio, "CIC Filter Introduction." For free publication by Iowegian, July 2000.

[18] K. Gentile, "Fundamentals of Digital Quadrature Modulation," *RF Design*, vol. RF Mixed Signal, pp. 40–47, February 2003.

[19] Komantech, "T/R40-16," 2001. Available: `http://www.komantech.com/product/Ultra_T_R40_16.htm` [Accessed 30 July 2008].

[20] Altera, "Quartus II Web Edition Software Version 8.0," 2008. Available: `http://www.altera.com/products/software/products/quartus2web/sof-quarwebmain.html` [Accessed 25 July 2008].

[21] National Semiconductor, *LF147/LF347, Wide Bandwdith Quad JFET Input Operational Amplifiers*, August 2000.

[22] M. Trninic, *Design and Construction of an Ultrasonic Radar Emulator*. Undergraduate Thesis, University of Cape Town, 2002.

[23] S. Korda, *Design and Construction of an Ultrasonic Radar Emulator*. Undergraduate Thesis, University of Cape Town, 2002.

[24] N. Lewitton, *Multiple Transducer Synthetic Apeture Sonar Interferometry for Emulating SAR Interferometry*. Master's Thesis, University of Cape Town, 2007.

[25] National Semiconductor, *LM3886 Overture - Audio Power Amplifier Series, High Performance 68W Audio Power Amplifier w/Mute*, October 2003.

[26] FERROXCUBE, *Product Selection Guide 2001*, 2001.

[27] Magnetics, *Ferrite Cores: Section 8 - RM Cores*, 2006. Rev. 1.

[28] Magnetics, *Ferrite Cores: Section 4 - Power Design*, 2006. Rev. 1.

[29] Mini-Circuits, *RF Transformers, Wideband*, 2007.

[30] National Semiconductor, *LM380 - 2.5W Audio Power Amplifier*, December 1972. Application Note 69.

[31] Murata, *Ultrasonic Sensors, MA40 Series*, 2001.

[32] R. H. Hosking, “Use FPGA resources to boost radar system perfromance,” *RF Design*, vol. Defense Electronics, pp. 16–19, October 2005.

[33] J. Baniak, D. G. Baker, A. M. Cunningham, and L. Martin, “Lockheed Martin Mission Systems: Silent Sentry Passive Surveillance,” June 1999.

[34] E. Nash, “IQ Modulators Advance Reconfigurable Radio,” *RF Design*, vol. Active Components, pp. 32–40, June 2006.

[35] S. Tao, T. Ran, W. Yue, and Z. Siyong, “Study of detection performance of passive bistatic radars based on FM broadcast,” *Journal of Systems Engineering and Electronics*, vol. 18, pp. 22–26, 2007.

[36] D. Shen, “Tutorial 6: Graph, Blocks and Connecting.” June 2005.

[37] T. Derham, S. Doughty, K. Woodbridge, and C. J. Baker, “Design and evaluation of a low-cost multistatic netted radar system,” *IET Radar Sonar Navig.*, vol. 1, pp. 362–368, 2007.

[38] V. Vitchev, “Mathematical Basics of Bandlimited Sampling and Aliasing,” *RF Design*, vol. Signal Processing, pp. 26–32, January 2005.

[39] Z. Jiabing, T. Liang, and H. Yi, “Study on moving target detection to passive radar based on FM broadcast transmitter,” *Journal of Systems Engineering and Electronics*, vol. 18, pp. 462–468, 2007.

[40] Airmar Technology Corporation, *Ultrasonic Transducers, Application Notes*, 2005.

[41] Ferrite Ceramics, *RM8/I, RM Cores and Accessories*, December 1999. (Datasheet).

[42] Phillips Components, *Soft Ferrites and Accessories*, April 2000.