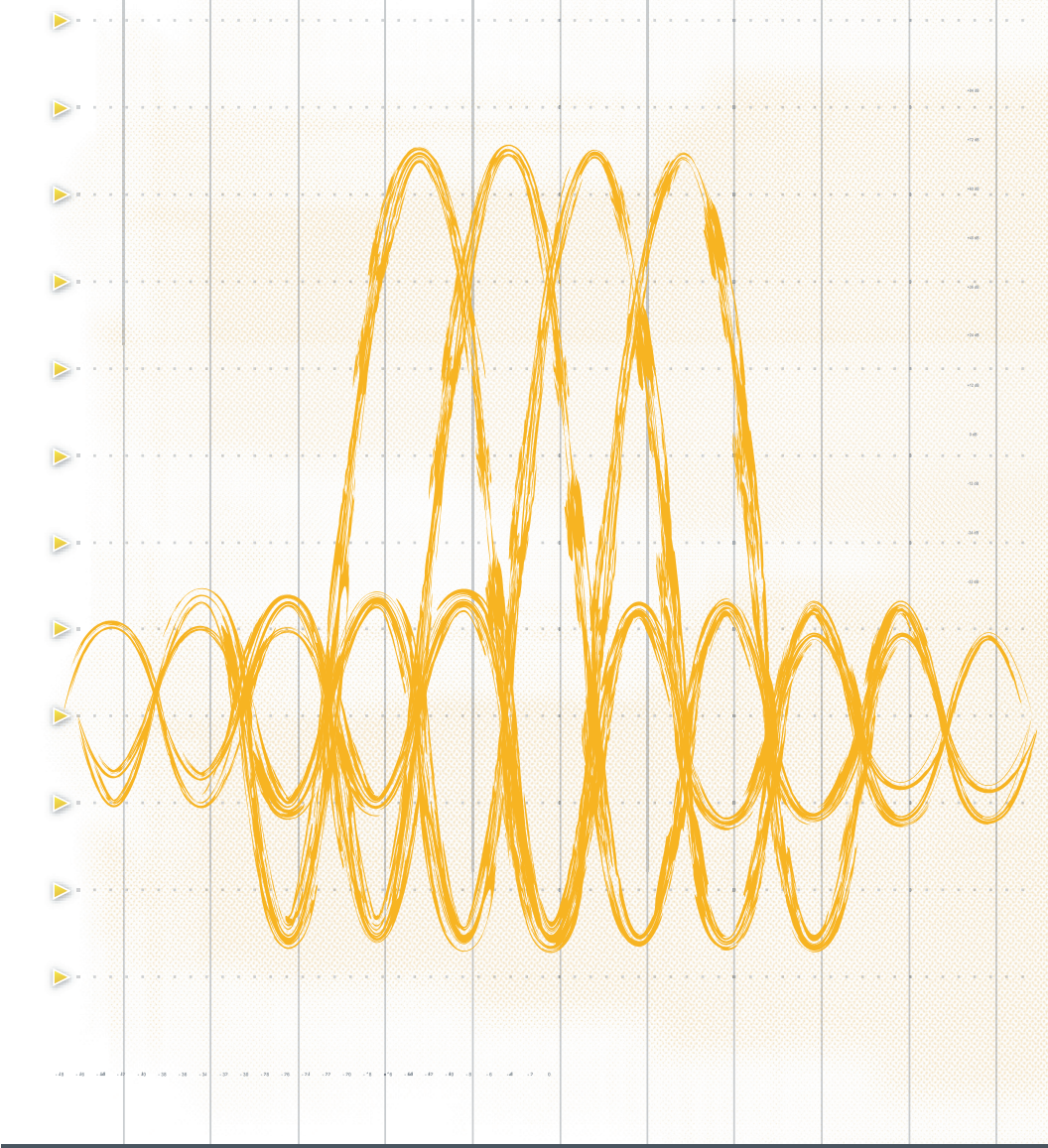

DIGITAL COMMUNICATIONS

PHYSICAL LAYER EXPLORATION LAB USING THE NI USRP™ PLATFORM

Robert W. Heath Jr., Ph.D., P.E.

Digital Wireless Communication

Physical Layer Exploration Lab Using the NI USRP

Student Lab Manual

Robert W. Heath Jr.

NATIONAL TECHNOLOGY & SCIENCE PRESS

ISBN-10: 1-934891-18-5
ISBN-13: 978-1-934891-18-6

10 9 8 7 6 5 4 3 2

Publisher: Tom Robbins
General Manager: Erik Luther
Technical Oversight: Sam Shearman
Marketing Manager: Amee Christian
Development Editor: Catherine Peacock
Compositor: Paul Mailhot, PreTeX Inc.

Dedication

I dedicate this to Garima, Pia, and Rohan for their love and support,

to my parents Bob and Judy Heath for their encouragement

and to Dr. Mary Bosworth for her passion in the pursuit of higher education.

R. W. H.

Contents

Preface

Wireless communication is fundamentally the art of communicating information without wires. In principle, wireless communication encompasses any number of techniques including underwater acoustic communication, semaphores, smoke signals, radio communication, and satellite communication, among others. The term was coined in the early days of radio, fell out of fashion for about fifty years, and was rediscovered during the cellular telephony revolution. Wireless now implies communication using electromagnetic waves—placing it squarely within the domain of electrical engineering.

Wireless communication techniques can be classified as either analog or digital. The first commercial systems were analog including AM radio, FM radio, television, and first generation cellular systems. Analog communication is rapidly being replaced with digital communication. The fundamental difference between the two is that in digital communication, the source is assumed to be digital. Modern applications of digital communication include cellular communication, wireless local area networking, personal area networking, and high-definition television.

Digital communication is widely taught in electrical engineering programs, at both the undergraduate and the graduate levels. Many digital communication courses take an abstract approach that emphasizes theory. This is no surprise—the mathematical theory of digital communication is beautiful and elegant. There has been a growing separation, however, in the abstract content of a typical course and what is implemented in practice. This difference is particularly evident in wireless communication systems, which have their own set of specific challenges. As courses are becoming more abstract, there is also increasing reliance on simulation to validate the theory. Typical simulations, though, are constructed under certain simplifying assumptions, which leaves important practical issues undiscovered. As a byproduct of the growing rift between theory and practice, and the increasing reliance on simulation, many students are not prepared to apply their knowledge immediately to a real digital wireless communication system.

This book is inspired by my own experiences as a graduate student and working at a startup company. My first summer project at Stanford University working with Arogyaswami Paulraj was to implement the physical layer of the receiver for a GSM system. The idea was to implement different space-time processing algorithms developed in Prof. Paulraj's Smart Antennas Research Group and to show practically how multiple antennas can improve system performance using real transmitted waveforms. I found the implementation of the space-time processing algorithms to be straightforward (not easy but for the most part following the theory). Most of my time was spent, however, dealing with more practical issues like channel estimation, frame synchronization, and carrier frequency synchronization. I found that the algorithms for dealing with these practical impairments were vitally important yet I had never encountered them in multiple courses on digital communication and wireless communication systems.

I used what I had learned later at a startup company then called Gigabit Wireless and later Iospan Wireless, to develop a prototype of a three transmit and three receive antenna MIMO (multiple input multiple output) communication system. This prototype was key to demonstrating the promise of MIMO communication, which in 1998 was not yet an established hot research area. Coincidentally this prototype used multichannel DAC and ADC products from National Instruments.

As a new Assistant Professor at The University of Texas at Austin, I longed to teach wireless communication from a practical perspective. I quickly found one reason why these concepts were not taught in the typical communication course—it was not included in most textbooks. As a result, I developed course notes that include these concepts and will be published in a forthcoming textbook [3]; the key concepts are included in this lab manual.

I did not just want to teach wireless communication including some theory about dealing with practical impairments. I wanted to teach a class where students would send and receive actual communication waveforms. Unfortunately, when I started developing the course materials in 2003 there were few viable solutions for the radio frequency components to create a repeatable and reliable system. All of this changed with the introduction of hardware platforms with flexible RF upconverters and downconverters, matched to high speed DACs and ADCs that facilitate the concept of software defined radio. Finally there was a way to create and process waveforms with high quality and reproduciabilty. Using such equipment, the students could focus on the algorithms and yet be exposed to practical wireless system engineering, without having to worry with the challenging analog front end and RF circuit design issues. The present book makes use of National Instrument's Universal Software Radio Peripheral (USRP) [13]. This low-cost hardware allows Universities to set up mul-

tiple workstations with a moderate budget. I am thrilled that this option has become available.

This book is a result of five years of development to make a complete laboratory course that teaches the principles of wireless digital communication. It consists of a series of labs with three components: a pre-lab, a laboratory experiment, and a lab report. The idea of the pre-lab is to write the essential LabVIEW code and getting it to work in a simulator before doing the laboratory experiment. This models conventional design practices where communication algorithms are tested in simulation before being tested over a wireless link. It is important to complete the pre-lab prior to starting the laboratory experiment. The laboratory experiments were designed to be completed in a time period of up to three hours. In the lab, the code from the pre-lab is run over the wireless link, some experiments are performed, and the results are recorded. The lab report is the final component of the lab. The purpose of the lab report is to discuss what was observed in the lab and to answer several questions related to wireless communication engineering. The lab report is an opportunity to synthesize what was learned.

Digital signal processing (DSP) is at the heart of the approach taken in this lab manual. No background in digital communication is assumed, though it would be helpful. The utility of a DSP approach is due to the bandlimited nature of wireless systems. Consequently with a high enough sampling rate, thanks to Nyquist's theorem, it is possible to represent the bandlimited continuous-time wireless channel from its samples. This allows the transmitted signal to be represented as a discrete-time sequence, the channel as a discrete-time linear time-invariant system, and the received signal as a discrete-time sequence.

The labs explore both single carrier and multicarrier transmission. Single carrier transmission uses quadrature amplitude modulation (QAM) and raised-cosine pulse-shaping. Over the course of the labs, complexity is added to the receiver design including functions like detection, channel estimation, equalization, frame synchronization, and carrier synchronization. In later labs, the system is extended to incorporate multicarrier modulation in the form of orthogonal frequency division multiplexing (OFDM) with coding.

This lab manual could not have been possible with the support from many graduate teaching assistants at The University of Texas at Austin including Roopsha Samanta, Sachin Dasnurkar, Ketan Mandke, Hoojin Lee, Josh Harguess, Caleb Lo, Robert Grant, Ramya Bhagavatula, Omar El Ayache, Harish Ganapathy, Zheng Li, and Tom Novlan. Some specific contributions deserve special mention. Roopsha Samanta developed the initial set of laboratory drafts and software in addition to being the first TA for the course. Ketan Mandke reworked the software to have a complete system implemented and a channel simulator, where components could be removed and replaced with

software from the student. He also revised the lab manual to reflect the new system implementation. Hoojin Lee undertook the job of creating figures for the early course material. Josh Harguess created a nice introduction to LabVIEW presentation. Robert Grant spent time seeing how to convert to an early version of the USRP platform, when it was just released. Caleb Lo, Ramya Bhagavatula, Omar El Ayache, and Tom Novlan all made important revisions to the notes and software to keep it up-to-date with current versions of LabVIEW. All the teaching assistants ran the labs, worked with the students, collected feedback, and make suggestions and revisions. I sincerely thank them for their support.

The development of this book and course was supported by several groups. National Instruments made the course possible through its establishment of the Truchard Wireless Lab at The University of Texas at Austin. NI provided summer support that funded the endeavor, and contributed countless hours of employee time. Several groups from University of Texas at Austin including the Department of Electrical Engineering and the Cockrell School of Engineering provided teaching assistant support to develop and maintain the course materials. Working on the labs has inspired my research. I am pleased to acknowledge research support from the National Science Foundation, DARPA, the Office of Naval Research, the Army Research Labs, Huawei, National Instruments, Samsung, Semiconductor Research Corporation, Freescale, Andrew, Intel, and Cisco. Their support has allowed me to train many graduate students who share my appreciation of theory and practice.

I hope you enjoy this book and doing the laboratory experiments.

ROBERT W. HEATH JR.
Austin, Texas

About the Author

Robert W. Heath Jr. is an Associate Professor in the Department of Electrical and Computer Engineering at The University of Texas at Austin and a member of the Wireless Networking and Communications Group. He is also President and CEO of MIMO Wireless Inc. and VP of Innovation at Kuma Signals LLC. He received his B.S. and M.S. degrees from the University of Virginia, Charlottesville, VA, in 1996 and 1997 respectively, and the Ph.D. from Stanford University, Stanford, CA, in 2002, all in electrical engineering. Prior to joining The University of Texas, he was with Iospan Wireless Inc, San Jose, CA where he worked on the design and implementation of the physical and link layers of the first commercial MIMO-OFDM communication system.

His current research interests include several aspects of wireless communication and signal processing including limited feedback techniques, multihop networking, multiuser and multicell MIMO, interference alignment, adaptive video transmission, manifold signal processing, and 60 GHz communication techniques. He has published some 300 papers in archival journals and conference proceedings on these topics, and has more than 25 U.S. patents.

He has been an Editor for the IEEE Transactions on Communication, an Associate Editor for the IEEE Transactions on Vehicular Technology, lead guest editor for an IEEE Journal on Selected Areas in Communications special issue on limited feedback communication, and lead guest editor for an IEEE Journal on Selected Topics in Signal Processing special issue on Heterogenous Networks. He currently serves on the steering committee for the IEEE Transactions on Wireless Communications. He was a member of the Signal Processing for Communications Technical Committee in the IEEE Signal Processing Society. Currently he is the Chair of the IEEE COMSOC Communications Technical Theory Committee. He was a technical co-chair for the 2007 Fall Vehicular Technology Conference, general chair of the 2008 Communication Theory Workshop, general co-chair, technical co-chair and co-organizer of the 2009 IEEE Signal Processing for Wireless Communications Workshop, local co-organizer for the 2009 IEEE CAMSAP

Conference, technical co-chair for the 2010 IEEE International Symposium on Information Theory, technical chair for the 2011 Asilomar Conference on Signals, Systems, and Computers, and is technical co-chair for the 2014 IEEE GLOBECOM conference.

He was a co-author of best student paper awards at IEEE VTC 2006 Spring, WPMC 2006, IEEE GLOBECOM 2006, IEEE VTC 2007 Spring, and IEEE RWS 2009. He is co-recipient of the Grand Prize in the 2008 WinTech WinCool Demo Contest and is co-recipient of the 2010 EURASIP Journal on Wireless Communications and Networking best paper award. He was a 2003 Frontiers in Education New Faculty Fellow. He is the recipient of the David and Doris Lybarger Endowed Faculty Fellowship in Engineering. He is a licensed Amateur Radio Operator and is a registered Professional Engineer in Texas. He is a Fellow of the IEEE.

Lab 1: Introduction to Wireless Communications Lab
Part 1: Introduction to NI LabVIEW

Summary

A software defined radio (SDR) is a communication radio that uses software to implement the algorithms necessary for digital communication. In this lab, you will be designing and implementing an SDR using National Instruments hardware and software (i.e., National Instruments LabVIEW). Through the implementation of this SDR you will investigate practical design issues in wireless digital communication.

This introductory lab is divided into two parts. In the first part you will learn about the software used in this lab; in the second part you will learn about the RF hardware to be used in lab. This part of the lab will answer the following questions:

- What software will I need for this lab?
- What is NI LabVIEW and why do we use it in this lab?
- What are some of the advanced LabVIEW tools used in this lab?
- Is there such a thing as good code style in LabVIEW?

Also, in this part of the lab, you will build some of the basic components of the SDR that you will be designing throughout the labs. This lab is meant to introduce LabVIEW as a tool for the design of digital communications systems in this course, and to give you a chance to get comfortable with basic LabVIEW programming concepts.

1 Software and Other Materials for the Course

Throughout the course you will be required to do some portion of the lab work (e.g., pre-labs) outside of the lab. This means that you will need to get access to the appropriate software tools and packages outside of class. The NI software below will be used in the course.

- NI LabVIEW 2010 or later
- NI LabVIEW Modulation Toolkit 4.3.1 or later

Note: Your instructor will provide information about how to access LabVIEW at your University.

2 Introduction to National Instruments LabVIEW

2.1 What Is LabVIEW?

LabVIEW is a graphical programming language developed by National Instruments. The basic building block of LabVIEW is the virtual instrument (VI). Conceptually, a VI is analogous to a procedure or function in conventional programming languages. Each VI consists of a *block diagram* and a *front panel*. The block diagram describes the functionality of the VI, while the front panel is a top level interface to the VI. The construct of the VI provides two important virtues of LabVIEW: code reuse and modularity. The graphical nature of LabVIEW provides another virtue: it allows developers to easily visualize the flow of data in their designs. NI calls this Graphical System Design. Also, since LabVIEW is a mature data flow programming language, it has a wealth of existing documentation, toolkits, and examples which can be leveraged in development.

In this course you will use National Instruments RF hardware. LabVIEW provides a simple interface for configuring and operating various external I/O, including the NI RF hardware used in lab. This is the main reason why you will use LabVIEW as the programming language to build an SDR in this course. You should realize that the algorithms considered here could also be programmed in optimized C/C++, assembly, or VHDL and implemented on a DSP, microcontroller, or an FPGA. The choice of hardware and software in this lab is mostly a matter of convenience.

In future labs you will need to be familiar with LabVIEW and the documentation/help available to you. This is the only lab in this course which will give you the opportunity to learn and practice LabVIEW programming; so it is important that you take this opportunity to ask the instructor any questions you might have about LabVIEW programming. The following tutorials and reference material will help guide you through the process of learning LabVIEW:

- LabVIEW 101 [11],
- *LabVIEW Fundamentals* from National Instruments [12],

- and online LabVIEW tutorials from NI [9, 10].

New LabVIEW programmers should carefully review all of the material in [11] and [9]. Please remember to refer to [12] and [10] often as they are excellent references for all basic LabVIEW questions. In order to test your familiarity with LabVIEW, you will be asked at the end of this lab to build some simple VIs in LabVIEW and integrate them into a simulator. Bring any questions or concerns regarding LabVIEW or these tutorials to your instructor's attention. For the remainder of this lab you should be familiar with the basics of LabVIEW programming and where to look for help.

2.2 What Advanced LabVIEW Tools Will I Use?

Many of the algorithms you will implement in the lab (and in digital communication in general) use linear algebra. LabVIEW provides a variety of support for matrix and vector manipulation and linear algebra. Some of the functions you will use regularly include matrix inversion (*Inverse Matrix.vi*), matrix multiplication (*A x B.vi*), and reshaping arrays (*Reshape Array.vi*). LabVIEW also has many built-in signal processing functions which will be used in the lab, such as the fast Fourier transform (*FFT.vi*), inverse FFT (*Inverse FFT.vi*), and convolution (*Convolution.vi*). The Modulation Toolkit is a toolset of common digital communication algorithms which will also be leveraged in this lab. Note that many of the functions you will implement in this lab are already available in the Modulation Toolkit in some form. The objective of this course is to understand the principles of wireless digital communication by implementing the physical layer in as much detail as possible. Once you are familiar with these concepts, you will decide when to use existing VIs and when to write your own. It is recommended that you explore the following tool palettes in order to get some exposure to the VIs that you will likely use in this course:

- Signal Processing palette
- Digital palette (part of the Modulation Toolkit palette)
- Structures palette
- Complex palette (part of the Numeric palette)
- and Array palette

Appendix A enumerates some common VIs to be used in this course and how to access them.

2.3 Code Style in LabVIEW

Googling the phrase "*labview code style*" will yield a number of sources for general code style in LabVIEW. Whatever style you choose to follow, remember that your code style should accomplish the following:

1. Consistency - All of your code should follow the same code style.
2. Readability - Your code should be user-friendly and easy to maintain.
3. Documented - Documentation should not be few and far between. Good documentation not only comes in the form of free text labels (see the Decorations palette, which is part of the Structures palette), but it also means good variable and function names.

When incrementally designing a moderately complicated system, it can be tempting to continually add *hacks* until you end up with *spaghetti code* (see Figure 1).

The design of a digital communication system is modular and sequential in nature. Thus it is natural to adopt a code style which makes use of a great deal of hierarchy and modularity. Since many algorithms in digital communications can often be boiled down to a recipe (*e.g., A then B then C*), it is critical to use sub-VIs whenever possible.

3 Pre-Lab

In this pre-lab you will build two sub-VIs: *source.vi* and *error_detect.vi*. You will then plug your code into a simulator provided to you (shown in Figure 2). The goals of this pre-lab are (1) to understand how to build and use sub-VIs and (2) to practice using good code style.

The details of each sub-VI that you must construct are given in Tables 1 and 2. In all future labs you will be provided with an appropriate simulator. You will typically reimplement certain blocks within the simulator and use it to perform some kind of measurement. This incremental process will continue throughout the course as you learn about each part of a digital communication system. The general simulator you will use in future labs is composed of many files. Figure 3 shows an overview of the hierarchy of the various parts of the simulator.

It should be noted that the simulator you will be provided with will have full functionality; however, the block diagram of the functional blocks of the simulator will be locked. This means that although you will be able to see how the simulator is supposed to function and look at the front panel of all the VIs, you will not be able to see the block diagram (i.e., the implementation) of the digital communication sub-VIs.

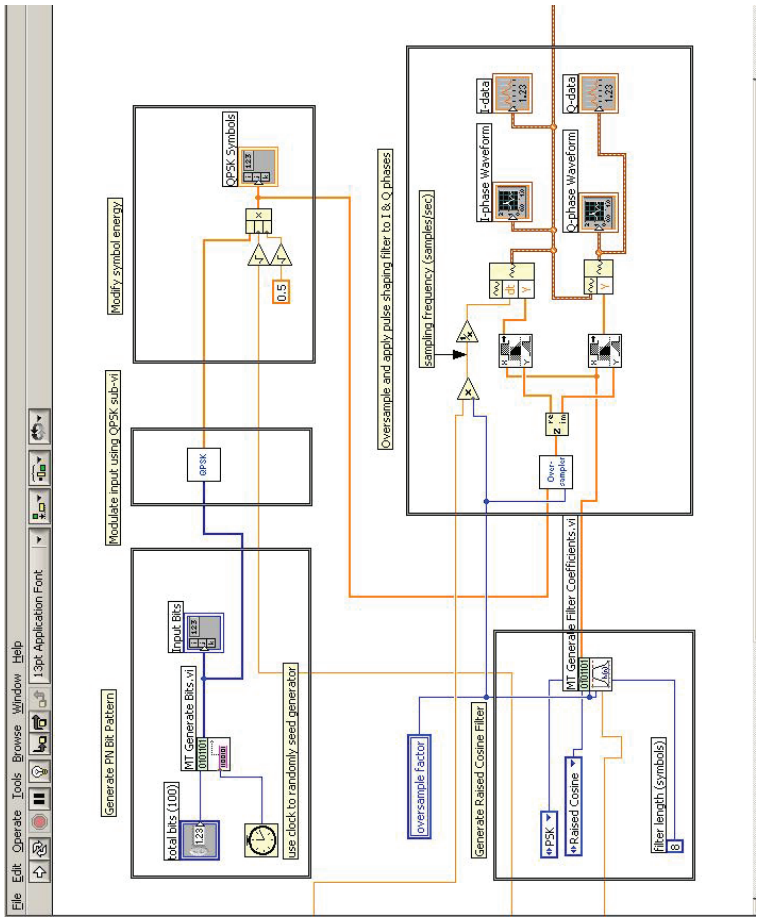

Figure 1: Spaghetti code design can be messy.

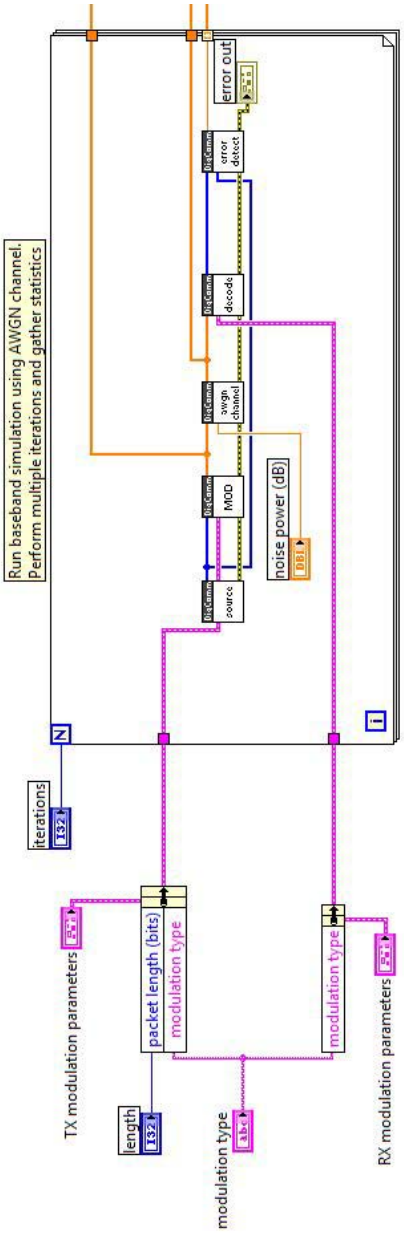

Figure 2: Digital communications simulator. Block diagram of *awgn_simple_sim.vi*.

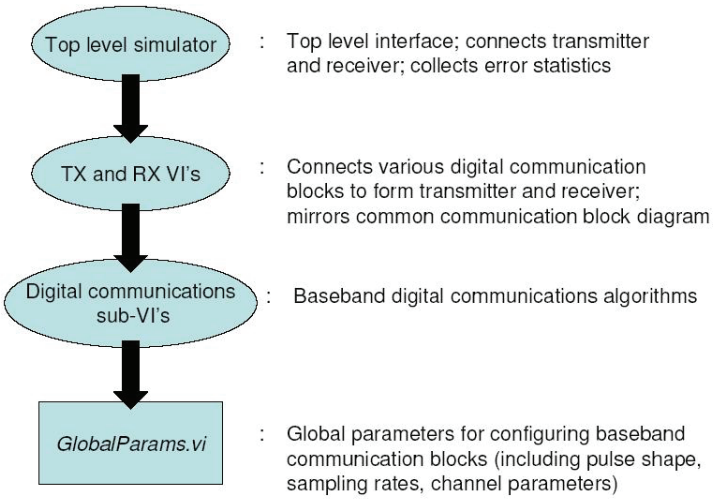

Figure 3: Overview of simulator hierarchy in LabVIEW.

Table 1: Description of *source.vi*

source.vi - generates a random bit sequence[1]			
INPUTS	*length*	I32 (32 bit integer)	length of bit sequence
OUTPUTS	*output bit sequence*	array of U8 (unsigned bytes)	random bit sequence (array of unsigned bytes taking value 0 or 1)

Once you have constructed the VIs for this lab (*source.vi* and *error_detect.vi*), be sure to appropriately edit the icon of each VI and connect all of the relevant terminals. Now you are ready to replace the corresponding blocks inside the simulator provided to you (shown in Figure 2). After placing your code in the simulator and verifying that it works correctly, observe what happens to the bit-error rate as you decrease the noise power in the simulator. (Note: If your subVI appears greyed out after placing it in the simulator, right click the subVI and choose "relink subVI" to fix this issue)

[1] *Hint: There are a number of ways to generate a random bit sequence in LabVIEW. For instance, you could do it using the random number generator provided in the Numeric palette, or you could just use the Generate Bits.vi module located in the Digital palette (see Modulation Toolkit palette).*

Table 2: Description of *error_detect.vi*

error_detect.vi - computes the bit error rate of a bit sequence			
INPUTS	*estimated bit sequence*	array of U8	estimated bit sequence
	key bit sequence	array of U8	correct bit sequence
OUTPUTS	*bit-error rate*	DBL (double precision)	Hamming distance[2] between estimated sequence and key sequence divided by length of bit sequence

The simulator will allow you to run multiple iterations and collect statistics from these runs. In this case, you will only be concerned with the bit-error rate (BER) statistics (i.e., the average BER).

Plot a curve of the average bit-error rate versus $1/N_0$ over a number of trials, where N_0 is the noise power (realize N_0 is given in dB in the simulator). It is sufficient to vary noise power from -10 dB to 0 dB in 2 dB steps. Use BPSK modulation and choose an input length of 200 bits. *Note that in order to observe a nonzero error for low noise powers you may need to run in excess of* 10^4 *iterations*.

HINT: (X/Y) *in dB is equal to* $10\log\left(\frac{X_{\text{linear}}}{Y_{\text{linear}}}\right)$, *where* $X_{dB} = 10\log(X_{\text{linear}})$ *and* $Y_{dB} = 10\log(Y_{\text{linear}})$. *The* $(\cdot)_{\text{linear}}$ *is a linear scale value, not dB.*

Pre-Lab Turn In

Submit your implementations of *source.vi* and *error_detect.vi*. Turn in your plots of average bit-error rate (BER) versus $1/N_0$.

A typical example of a bit-error-rate plot is shown in Figure 4. You may use LabVIEW, Excel, or any other graphing utility to generate these plots (i.e., you only need to use LabVIEW to generate the data, not necessarily to generate the plots).

[2]The Hamming distance of two bit sequences is equal to the number of positions in which the sequences differ.

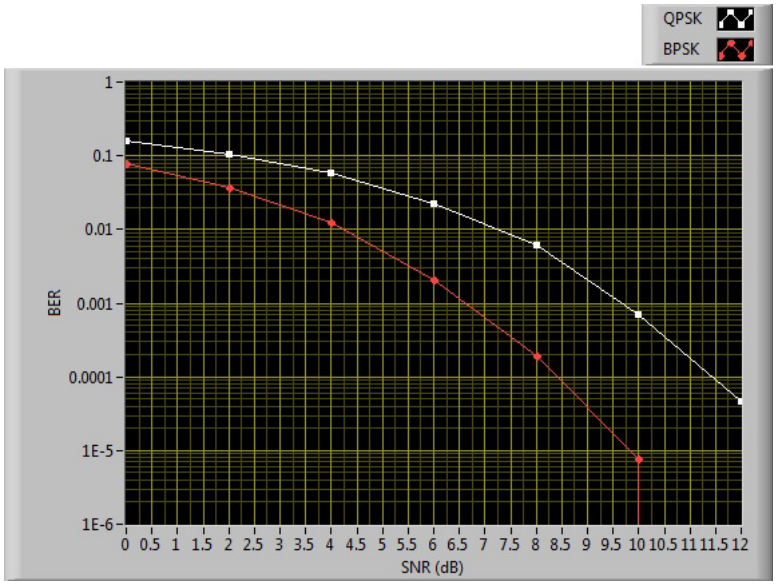

Figure 4: BER plot for BPSK and QPSK modulation in AWGN channel.

4 Lab Experiment

There is no in-lab component for this lab.

5 Lab Report

After most labs you will write a lab report. You will only turn in one lab report per lab. This means that if a lab has multiple parts (e.g., Parts 1 and 2 of Lab 1), you will only be required to turn in one lab report. The contents of this report will be explicitly stated in the lab draft. *Note: This means that you will not be submitting a lab report for this part of Lab 1.* Upon completion of Part 2 of Lab 1, you will be required to generate a lab report.

Lab 1: Introduction to Wireless Communications Lab
Part 2: Introduction to NI RF Hardware

Summary

In this part of the introductory lab you will learn about the NI RF hardware to be used in the Wireless Communications Lab. You will acquaint yourself with the relevant documentation and references available to you with regard to the RF hardware. You will build some simple example VI's from the NI documentation and use some of the existing LabVIEW tools (e.g., RF Signal Analyzer Demo Panel). In the final part of this lab you will explore some of the VIs that you will use in future labs to control the RF hardware. This part of Lab 1 introduces NI RF hardware as a tool for digital communication, and gets you familiar with the help/documentation available for the hardware you will be working with throughout the course.

In this lab, for the pre-lab, you are required to turn in only the questions given in Section 2. There are no pre-lab VIs for this lab. As a part of the lab session, you will construct two VIs by following the instructions given in the NI documentation. The questions given in Section 3 need to be answered during the lab only. Please do not submit the answers to these questions with your pre-lab. At the end of this lab, you will submit a lab report, the details of which are given at the end of this document.

1 RF Hardware in Wireless Communications Lab

In this course you will use software recofigurable RF hardware from National Instruments to build a digital communication system. This hardware can be easily configured using LabVIEW. The RF Hardware used in the lab is the National Instruments USRP (Universal Software Radio Peripheral). Figure 1 shows a USRP connected to a PC (running LabVIEW). This PC controls the USRP through the gigabit ethernet cable connecting the two together. In this section you will learn more about these modules.

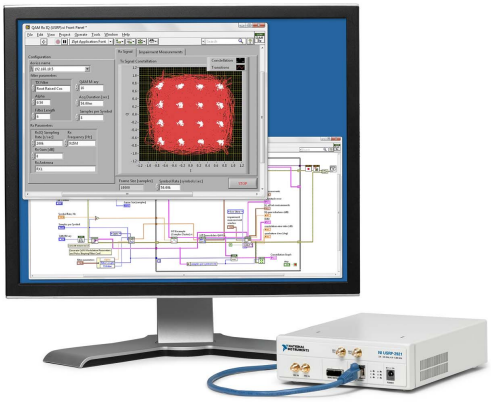

Figure 1: Hardware setup in Wireless Communications Lab.

1.1 NI USRP (Transmit)

Let's begin by taking a look at the NI USRP when acting as a transmitter. The USRP receives a waveform from the host PC with 16 bits of resolution sampled at up to 25 MSamples/second. This signal is upconverted to a radio frequency (RF) before being sent to an amplifier and then transmitted over the air. For more information on the NI USRP, please refer to the help files associated with the USRP (**Help⇒NI-USRP Help⇒Devices⇒NI USRP-292x Specifications**)

1.2 NI USRP (Receive)

The NI USRP is also capable of receiving, and does so in the following manner. The received signal is mixed with a desired carrier frequency in order to down-convert it to a complex IQ baseband signal sampled at 100 MSamples/second. The digital signal is then downsampled to a rate specified by the user and passed to the host PC for processing.

When the ADC of the NI USRP samples at the full digitizer rate (100 MSamp/sec), it can acquire a bandwidth of 20 MHz. Sampling at such a high rate with high resolution (14 bits) produces a large amount of data. However, to acquire a signal with smaller bandwidth (a narrowband signal), it is sufficient to sample at a rate twice that of the signal bandwidth[1].

[1]This sampling rate is known as the Nyquist rate of the system. It is the minimum sampling rate required to reproduce a bandlimited signal from discrete samples.

Decimation is the process of reducing the sampling rate of a discrete-time signal. In practice, this entails applying an anti-aliasing filter and throwing away some samples. The NI USRP features a digital down converter (DDC), which allows it to perform decimation in hardware instead of in software (which can be considerably slower).

Using the DDC, the USRP can acquire narrowband signals at sample rates much less than the full digitizer rate (100 MSamp/sec), thus reducing the memory required to store a waveform. For example, a waveform sampled at 100 MSamp/sec for 1 second produces 100 times more data than the same waveform sampled at 1 MSamp/sec. The effective sample rate of the USRP (including DDC) ranges from a maximum of 25 MSamp/sec down to 200 kSamp/sec and is directly configurable by the user. For more information on the NI USRP, please refer to the associated help files.

2 Pre-Lab

Review the documentation for the USRP (**Help⇒NI-USRP Help**). You will find that the help files found under **Devices⇒NI USRP-292x Specifications** and **Devices⇒ NI USRP-2920** contain the most comprehensive information about the USRP. For a deeper understanding of the specifications, please refer to the following documents:

- Bandwidth, Sample Rate and Nyquist Theorem [8]
- Understanding RF & Microwave Specifications - Part I [14]
- Understanding RF & Microwave Specifications - Part II [15]

Answer the following questions about the USRP after reading through the help files.

1. What is the range of allowable carrier/center frequency supported by the NI-USRP?
2. What is the maximum allowable bandwidth supported by the NI-USRP?
3. What is the maximum sampling rate of the NI-USRP?
4. Why do you think the DDC is implemented? What is its main benefit?

Answer the following general concept questions using [8], [14] and [15].

1. In your own words, describe what the bandwidth of an instrument is.

2. What is meant by the sampling rate of an instrument?

3. Why are these specifications important for designing a transmitter and receiver in a wireless communications system?

Pre-Lab Turn In

Submit your answers for all of the above pre-lab questions. As a reminder, this is all that you need to turn in for the pre-lab. There are no pre-lab VIs to be submitted for this lab.

3 Lab Experiment

3.1 Building a Basic Sine Generation VI

For the first task in this lab you will build a basic sine generation VI described below. After you have completed building this example VI, verify that the block diagram of your VI resembles Figure 2, and that the front panel resembles Figure 3.

Use the following procedure to begin using the NI USRP to generate a signal:

1. Select **Start⇒Programs⇒National Instruments⇒LabVIEW 2010⇒LabVIEW** to launch LabVIEW. The LabVIEW dialog appears.

2. Click **New**, choose **Blank VI**, and click **OK** to create a blank VI.

3. Display the block diagram by clicking it or selecting **Window⇒Show Block Diagram**.

4. Navigate to the NI USRP VIs on the **Functions⇒Instrument I/O⇒ Instrument Drivers⇒NI-USRP⇒TX** palette. With LabVIEW running, click here to find the **NI USRP TX** LabVIEW palette.

5. Create the block diagram shown in Figure 2 by placing the four core VIs on the block diagram in the order they appear in the NI USRP RX functions palette.

6. Hover the cursor over the **device name** terminal on the niUSRP Open TX Session VI and right-click. Select **Create⇒Control** to create a front panel field where you specify the NI USRP device name.

7. Hover the mouse tool over the **IQ rate**, **carrier frequency**, and **gain** terminals of the niUSRP Configure Signal VI.

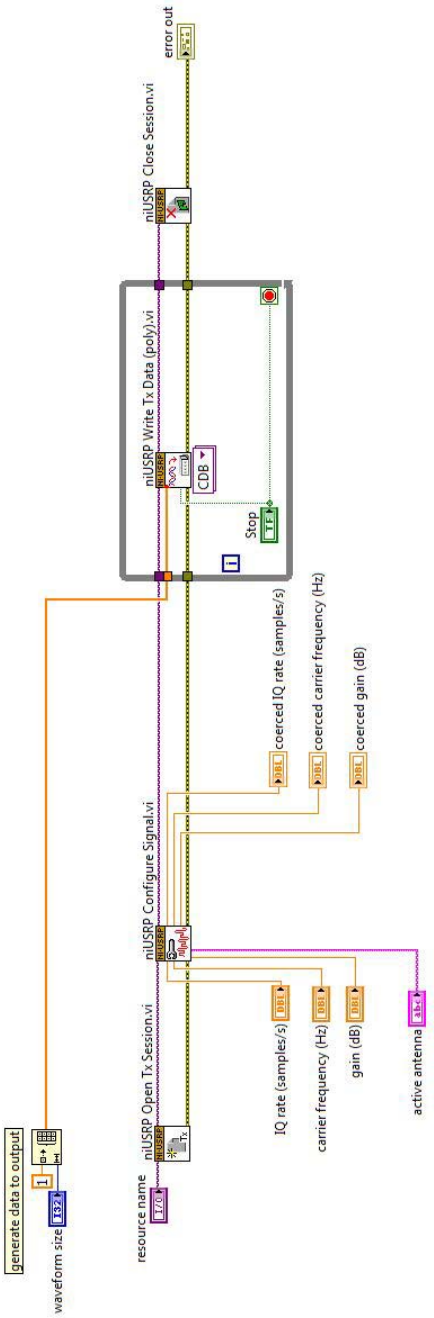

Figure 2: Block diagram of *Basic Sine Wave Generation VI*.

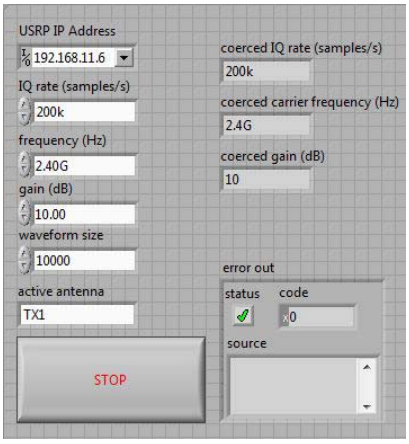

Figure 3: Front panel of *Basic Sine Wave Generation VI*.

8. Right-click each terminal and select **Create⇒Control** from the shortcut menu to create frequency, IQ rate, and gain controls.

9. Display the front panel by clicking it or selecting **Window⇒Show Front Panel**. Fields are displayed in which you can specify a frequency, IQ rate, and gain.

Continuous waveform generation is controlled by means of a STOP button. A STOP button is typically used within a While Loop. This example places a While Loop around the Write TX Data VI so that signal generation continues until you click STOP.

Build a While Loop and STOP button by completing the following steps:

1. Display the block diagram by clicking it or selecting **Window⇒Show Block Diagram**.

2. Select the While Loop on the **All Functions⇒Structures** palette. With LabVIEW running, click here to find the While Loop on the **Structures** palette.

3. Enclose the niUSRP Write TX Data (Poly) VI in the While Loop.

4. Hover the mouse tool over the **Loop Condition** terminal of the While Loop.

5. Right-click the **Loop Condition** terminal and select **Create Control** from the shortcut menu to create a **STOP** button on the VI front panel.

6. Create an error indicator by right-clicking on the **error out** terminal of the niUSRP Close Session VI and selecting **Create⇒Indicator**.

Now we need to produce a waveform to be transmitted. We will be transmitting a sinusoid at a carrier frequency that is specified by the **carrier frequency** control. This means that the baseband signal we will be passing to the NI USRP is simply a constant signal. To generate this signal, complete the following steps:

1. Using the quick-drop functionality described in Lab 1, Part 1, place the **Initialize Array** function down on the block diagram.
2. Right-click the **element** input of the **Initialize Array** function and select **Create⇒Constant**. Double-click on the constant that was created and change it to 1.
3. Right-click on the **dimension size** input and select **Create⇒Control**.
4. Wire the output of the **Initialize Array** function block to the **data** input of the niUSRP Write Tx Data (poly) VI.
5. Finally, wire the STOP button that was created for the While Loop to the **end of data?** input of the niUSRP Write Tx Data (poly) VI and select **CDB** from that VIs drop down menu.

Remember that in order to operate your sine generation VI, you must properly configure the *USRP IP Address* for the USRP (an input to *niUSRP Initialize.vi*). You can find the IP addresses for all of the USRP devices connected to your PC by using the **NI-USRP Configuration Utility**. Once that is properly configured, set the rest of the parameters as follows and run the VI:

- carrier frequency = 2.4 GHz
- IQ rate (samples/s) = 400k
- gain (dB) = 20
- waveform size = 10000
- active antenna = TX1

You will now test your VI using the niUSRP EX Spectral Monitoring (Interactive) VI, which can be found in the Functions palette. This spectrum analyzer tool will allow you to observe the frequency domain response of your basic sine wave transmitted over a wireless link. The following directions describe the basic operation of the Spectral Monitoring VI.

1. First, open the Spectral Monitoring VI. This can be accessed from **Functions Palette⇒Instrument I/O⇒Instrument Drivers⇒ NI-USRP⇒Examples⇒niUSRP EX Spectral Monitoring (Interactive).vi**.

2. Verify that the USRP IP Address match the values specified in the NI-USRP Configuration Utility.

3. After opening the example VI, you will want to modify the hardware settings on the front panel. Use the parameters listed below for this experiment.

   - Center Frequency = 2.4 GHz
   - IQ Sampling Rate [s/sec] = 400k
   - Acq Duration [sec] = 50m

Questions

Answer the following questions regarding the NI USRP.

1. What is the IP Address for your NI USRP?

2. Describe what happens to the spectral response of the transmitted signal when you place an obstruction between the antenna of transmitter and that of the receiver (i.e., blocking the line of sight component of the waveform).

Turn In

Demonstrate to the instructor that your basic sine generation VI is working properly. You will need to start the Spectral Monitoring VI and show that your basic sine generation VI produces a carrier signal at 2.4 GHz.

Submit your answers to the above questions in your lab report. More details about the requirements for the lab report are discussed near the end of this document.

3.2 Programming the RF Signal Analyzer

In the second part of this lab experiment you will build a Continuous IQ Acquisition VI and it should look like the Continuous IQ Acquisition VI shown in Figure 4. Add all of the appropriate controls and indicators necessary to control the sub-VIs in your code.

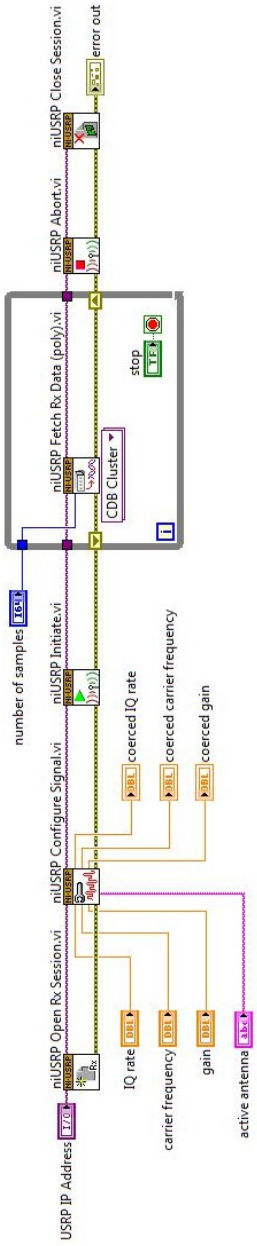

Figure 4: Continuous IQ Acquisition VI.

To test your acquisition VI, you will graph the real and complex parts of the signal acquired by *niUSRP Fetch Rx Data (poly).vi*. To do this:

1. Place two waveform graphs on the front panel of your VI. These can be accessed from the **All Controls** tool palette (**All Controls⇒Graph⇒ Waveform Graph.vi**).
2. Retrieve the real and imaginary waveform components from the *IQ Waveform* output of *niUSRP Fetch Rx Data (poly).vi*. Use *Get Complex IQ Component.vi* to retrieve these components from *IQ Waveform*. This VI can be accessed from the **All Functions** tool palette on the block diagram (**All Functions⇒Modulation⇒Analog⇒Utilities⇒Get Complex IQ Component.vi**).
3. On the block diagram, connect the terminal for each waveform graph to its corresponding waveform for the real and imaginary parts of the acquired signal (inside of the While-Loop structure).

You will use the basic sine generation VI created in the previous section to test your acquisition VI. In order to do so, you must first set up the parameters for the receiving USRP. Set the input parameters for *niUSRP Configure Signal.vi* listed below. Notice the carrier signal generated by your basic sine generation VI will be offset from the center frequency of your acquisition VI, effectively creating a 100 kHz single-sideband (SSB) modulated waveform at baseband.

- Carrier Frequency = {2.4001 GHz @ transmitter} and {2.4 GHz @ receiver}
- IQ rate = 800k
- gain = 20 dB
- number of samples = 10000
- active antenna = RX1

You should now be ready to continuously acquire a signal. After setting the parameters described above, run both VIs and observe the in-phase and quadrature-phase waveforms (i.e., real and imaginary parts respectively) of the acquired signal.

Questions

Answer the following question about the NI USRP.

1. What are the three elements in the *data* cluster output from *niUSRP Fetch Rx Data (poly).vi*? Give the name and data type of each element.

Turn In

Demonstrate to the instructor that your acquisition VI is working properly. It must correctly display both I and Q phases of the acquired signal.

Submit your answers to the questions in your lab report.

3.3 Looking Ahead

For the rest of this course, VIs for RF hardware configuration and operation will be provided to you. This will allow you to focus on the details of the digital communications algorithms you will be implementing. Even though you will be provided with the VIs necessary for controlling the RF hardware, you will be expected to understand what these VIs do and how they work.

You will be using the VIs listed below in lab to configure and operate the RF hardware. Explore the front panels and block diagrams of these VIs.

- *TXRF_init.vi* - initializes a USRP transmit session with the appropriate RF parameters.
- *TXRF_send.vi* - executes a transmit operation (i.e., sends a complex waveform), and closes the USRP session.
- *RXRF_init.vi* - initializes an USRP receive session with the appropriate RF parameters.
- *RXRF_recv.vi* - acquires a complex waveform, and closes the USRP session.

Questions

Answer the following questions about the VIs you will be using in lab. *Hint: The help files for VIs can be accessed from the block diagram. Right-click on the VI you wish to find out more about and select "Help."*

1. In *RXRF_config.vi* the **IQ rate** parameter is used by *niUSRP Configure Signal.vi*. Can the USRP support any arbitrary IQ rate? If not, what will the USRP do when an unsupported bandwidth parameter is requested?

Turn In

Submit all of your answers in your lab report.

DIRECTIONS FOR SUBMISSION OF THE LAB REPORT

Your lab report should accomplish the following tasks.

1. Answer all of the questions asked in lab (i.e., do not reanswer pre-lab questions).
2. Discuss problems you encountered in Lab 1, Part 1 and how you overcame these obstacles.
3. In 2-3 sentences describe the basic operation of the NI USRP when transmitting (e.g., what are the inputs/outputs for the USRP?).
4. In 2-3 sentences describe the basic operation of the NI USRP when receiving (e.g., what are the inputs/outputs for the USRP?).
5. Discuss problems you encountered in this part of Lab 1 and how you overcame these obstacles.

Lab 2: Baseband QAM Modem
Part 1: Modulation and Detection

Summary

In Lab 2, you will build a baseband digital modem that can use binary phase-shift keying (BPSK) or quadrature phase-shift keying (QPSK) modulation. The complexity of the transmitter and the receiver will grow in subsequent labs as more kinds of receiver impairments are considered. The main concepts in this lab include constellation mapping, oversampling, pulse shaping, matched filtering, maximum-likelihood detection, and probability of error calculation.

The system considered in this lab is illustrated in Figure 1. This modem transmits symbols and applies a detection algorithm assuming only an additive white Gaussian noise (AWGN) channel. The transmitter maps bits to elements of a symbol constellation. The sequence of symbols is upsampled then filtered in discrete-time by a transmit pulse shape. The filtered sequence

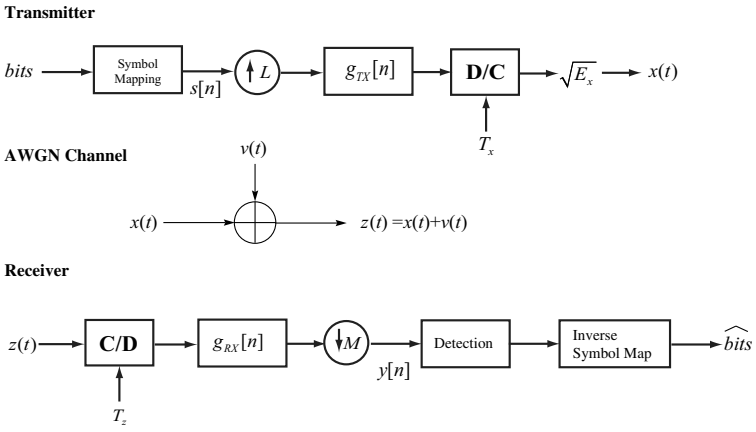

Figure 1: The system under consideration in this lab. The transmitter uses digital pulse-shaping and upsampling to create the transmit waveform. The baseband signal encounters additive white Gaussian noise during transmission. The receiver uses digital pulse-shaping and downsampling, followed by detection, to find a good guess of the transmitted symbols.

is then passed to the discrete-to-continuous converter. The receiver samples the received signal and filters it with the receiver pulse shaping filter. The filtered symbols are passed to a detection block which determines the most likely transmitted symbol for that observation. The detected symbols are passed to an inverse symbol mapping block to produce a good guess of the transmitted bits.

This lab is broken up into two parts. In this first part, you will build the modulation and decoding blocks of the modem in LabVIEW. After verifying the correctness of your design using a simulator, you will implement your code in lab using RF Hardware to see how your code works over a real wireless link. The goal of this part of Lab 2 is to introduce the basic concepts of modulation and detection. In the second part, you will implement a pulse-shaping filter (on the transmitter side) and the matched filter (on the receiver side).

You will be required to turn in two VIs (*modulate.vi* and *demodulate.vi*), plot the BER curves for QPSK and BPSK, and answer the questions in Section 2, as a part of the pre-lab submission. The lab report for this part of the lab will need to be submitted with the lab report in Part 2 of Lab 2 (i.e., there is no lab report required for this part of the lab).

1 Background

This section provides some review of the concepts of digital modulation and maximum likelihood detection. The emphasis is on BPSK and QPSK modulation, with maximum likelihood detection assuming AWGN.

1.1 Modulation

Digital modulation consists of mapping a sequence of bits to a (finite) set of analog waveforms which are suitable for transmission over a particular channel. There are many different kinds modulation. This and subsequent labs focus on what is known as complex pulse amplitude modulation. With this modulation technique, symbols are modulated onto pulse-shapes.

The source for digital modulation is a sequence of bits $\{b[n]\}$. The bit sequence is passed to the physical layer of a radio by higher layers.

The sequence of bits is processed by the symbol mapping block to produce a sequence of symbols $\{s[n]\}$. Essentially, each value $s[n]$ is a complex number that comes from a finite set of symbols called the *constellation* and written as

$$\mathcal{C} = \{s_0, ..., s_{M-2}, s_{M-1}\}.$$

The entries of the constellation are different (possibly complex) values. The size of the constellation, or cardinality, is denoted $|\mathcal{C}| = M$ where M is the number of symbols in the constellation. For practical implementation $M = 2^b$ where b is the number of bits per symbol so that a group of b input bits can be mapped to one symbol.

This lab considers two of the most common modulations found in commercial wireless systems.

- Binary Phase Shift Keying (BPSK) has

  $$\mathcal{C} = \{+1, -1\}.$$

  This is arguably the simplest constellation in use. The bit labeling is provided in Table 1. BPSK is a real constellation since it does not contain an imaginary component.

- Quadrature phase shift keying (QPSK) is a complex generalization of BPSK with

  $$\mathcal{C} = \{1 + j, -1 + j, -1 - j, 1 - j, \}.$$

  Essentially QPSK uses BPSK for the real and BPSK for the imaginary component. QPSK is also known as 4-QAM. The bit labeling is provided in Table 1.

For implementation and analysis, it is convenient to normalize the constellation to have unit energy. This means scaling the constellation such that

$$\frac{1}{M}\sum_{m=0}^{M-1} |s_m|^2 = 1.$$

The normalized QPSK constellation is illustrated in Figure 2.

The symbols are pulse-shaped and scaled by $\sqrt{E_x}$ to produce the complex baseband signal

$$x(t) \quad = \quad \sqrt{E_x}\sum_{n=-\infty}^{\infty} s[n]g_{\text{tx}}(t - nT). \tag{1}$$

The scaling factor E_x is used to model the effect of adding energy or power to $x(t)$. The scaling factor will be added typically in the RF by controlling the gain of the transmit power amplifier. The pulse-shape is given by the function $g_{\text{tx}}(t)$. To preserve energy, the pulse-shaping function is assumed to be normalized such that $\int_{-\infty}^{\infty} |g(t)|^2 dt = 1$. The symbol period is given by T. The symbol rate is $1/T$.

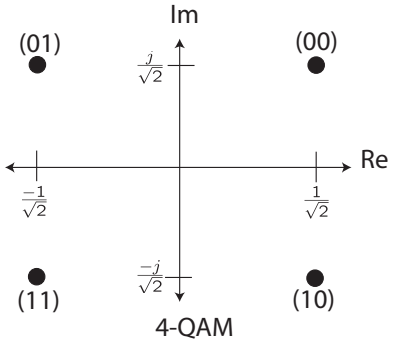

Figure 2: QPSK or 4-QAM constellation with generally accepted bit to symbol mappings based on Gray labeling.

Input Bit	Symbol
0	1
1	-1

(a)

Input Bits	Symbol
00	$1+j$
10	$-1+j$
11	$-1-j$
01	$1-j$

(b)

Table 1: Symbol Constellation Mappings for (a) BPSK and (b) QPSK

We refer to $x(t)$ as a complex pulse amplitude signal because complex symbols modulate a succession of pulses given by the function $g_{\text{tx}}(t)$. Effectively, symbol $s[n]$ rides the pulse $g_{\text{tx}}(t-nT)$. The symbol rate corresponding to $x(t)$ in Eq. (1) is $1/T$. The units are symbols per second. Typically in wireless systems this is measured in kilo-symbols per second or mega-symbols per second. Be careful to note that $1/T$ is not necessarily the bandwidth of $x(t)$, which depends on $g_{\text{tx}}(t)$. The bit rate is b/T (where recall there are 2^b symbols in the constellation) and is measured in bits per second. The bit rate is a measure of the number of bits per second the baseband waveform $x(t)$ carries.

Complex baseband signals are used for notational convenience. The signal $x(t)$ is complex because the resulting wireless signal is upconverted to "ride" on a carrier frequency. After upconversion, the resulting passband

signal will be $x_p(t) = \text{Re}\{x(t)\}\cos(2\pi f_c t) - \text{Im}\{x(t)\}\sin(2\pi f_c t)$. Up-conversion is performed in the analog hardware. The signal $x_p(t)$ will be launched from the transmit antennas into the propagation environment.

For practical implementation, the pulse-shaping is performed in discrete-time. One way to implement the pulse-shaping is illustrated in Figure 1. The idea is to use a discrete-to-continuous converter operating with a sample period of $T_x = T/L$ where $L > 0$ is an integer that denotes the oversampling factor. The symbol sequence is upsampled by L and then filtered by the oversampled pulse-shaping filter defined by $g_{\text{tx}}[n] := g_{\text{tx}}(nT_x)$.

1.2 Detection at the Receiver

When a signal propagates through the wireless channel, it will be affected by noise and other types of channel distortion. They type of distortion impacts the architecture of the receiver and the design of the signal processing algorithms. In this lab, it is assumed that the transmitted signal is only impaired by AWGN. This assumption will gradually be relaxed in future labs.

The AWGN communication channel is a good model for the impairments due to thermal noise present in any wireless communication system. Mathematically

$$z(t) = x(t) + v(t),$$

where $x(t)$ is the complex baseband signal, $v(t)$ is the AWGN, and $z(t)$ is the observation. The assumption that $v(t)$ is AWGN means the noise is an independent and identically distributed (i.i.d.) complex Gaussian random variable. In the presence of thermal noise, the total variance is $\sigma_v^2 = N_o = kT_e$, where k is Boltzman's constant $k = 1.38 \times 10^{-23}\text{J/K}$ and the effective noise temperature of the device is T_e in Kelvins. The effective noise temperature is a function of the ambient temperature, the type of antennas, as well as the material properties of the analog front end.

Under the assumed transmitted structure, the optimum receiver structure involves matched filtering, sampling at the symbol rate, and detection. The structure that will be implemented in this lab is illustrated in Figure 1. The received signal $z(t)$, which is assumed to be bandlimited, is sampled with a sampling rate of $1/T_z$. The sampling rate should be such that Nyquist is satisfied. In this lab we assume that $T_z = T/M$ where M is the integer oversampling factor. The received signal is filtered by $g_{\text{rx}}[n] = g_{\text{rx}}(nT_z)$ where $g_{\text{rx}}(t) = g_{\text{tx}}(-t)$ is the matched filter. After the digital filtering, the signal is downsampled by M to produce the symbol-rate sampled signal given by $y[n]$. The digital filtering is performed such that $y[n]$ is the equivalent to $y(t) = \int_t g_{\text{rx}}(\tau)z(t-\tau)d\tau$ sampled at nT.

Under assumptions about the transmit and receive pulse-shapes, described in more detail in Part 2 of Lab 2, a model for the received signal is

$$y[n] \quad = \quad \sqrt{E_x}s[n] + v[n], \tag{2}$$

where $v[n]$ is i.i.d. complex Gaussian noise with $\mathcal{N}_c(0, \sigma_v^2)$ where $\sigma_v^2 = N_o$ for complex thermal AWGN. More background on matched filters and pulse-shapes is provided in Part 2 of Lab 2.

A symbol detector is an algorithm that given the observation $y[n]$ produces the best $\hat{s}[n] \in \mathcal{C}$ according to some criterion. In this lab we consider the maximum likelihood (ML) criterion which solves

$$\hat{s}[n] = \arg\max_{s \in \mathcal{C}} f_{y|s}(y[n]|s[n] = s), \tag{3}$$

where $f_{y|s}(\cdot)$ is the conditional distribution of $y[n]$ given $s[n]$ and is known as the likelihood function. For additive white Gaussian noise, the conditional probability has a simple form. This allows the detection problem in Eq. (3) to be simplified to

$$\hat{s}[n] = \arg\min_{s \in \mathcal{C}} \ln f_{y|s}(y[n]|s[n] = s) = \arg\min_{s \in \mathcal{C}} |y[n] - \sqrt{E_x}s|^2. \tag{4}$$

Essentially the algorithm works as follows. Given an observation $y[n]$, it determines the transmitted symbol $s \in \mathcal{C}$, scaled by $\sqrt{E_x}$, that is closest to $y[n]$ in terms of the euclidean distance in the error term $y[n] - \sqrt{E_x}s$. The algorithm in Eq. (4) can be simplified by exploiting structure and symmetry in the constellation.

Note: If x is a complex symbol, then $\|x\|^2 = x_{\text{real}}^2 + x_{\text{imag}}^2$, where x_{real} and x_{imag} are the real and imaginary parts of x respectively.

2 Pre-Lab

To prepare for this part of the lab you will be required to build the modulation and decoding blocks of Figure 1. Tables 2 and 3 describe the details of the VIs you must implement. Plug your code into transmitter.vi and receiver.vi and run Simulator.vi (see the VI hierarchy in Figure 3) to verify functionality and correctness. Do not use the Modulation Toolkit (MT) VIs to implement these two VIs (i.e., do not just create a wrapper around an MT VI).

Note: From this point forward, please uniquely name your VIs so that they do not exactly match those of the VIs in the simulator (i.e., instead of modulate.vi, use yourname_modulate.vi).

Note that both *modulate.vi* and *decode.vi* do not have an input to specify modulation type. Modulation type is a parameter that can be found bundled in the Modulation Parameters cluster, which has already been inserted into the template VIs for you.

Table 2: Description of *modulate.vi*

modulate.vi - maps bit sequence to symbols using appropriate constellation; modifies symbol energy of output symbol sequence			
INPUTS	*input bit sequence*	array of U8 (unsigned bytes)	input bit sequence (each array element takes on value 0 or 1)
	modulation type	string	determines type of constellation map (i.e., BPSK or QPSK)
	Symbol Energy	double	determines the per-symbol energy of the output sequence (default = 1)
OUTPUTS	*output symbols*	array of CDB (complex doubles)	appropriately modulated output symbols

Table 3: Description of *decode.vi*

decode.vi - performs ML detection on a given symbol sequence			
INPUTS	*input symbols*	array of CDB (complex doubles)	input symbols to be decoded
OUTPUTS	*estimated bit sequence*	array of U8 (unsigned bytes)	decoded output bit sequence

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break.

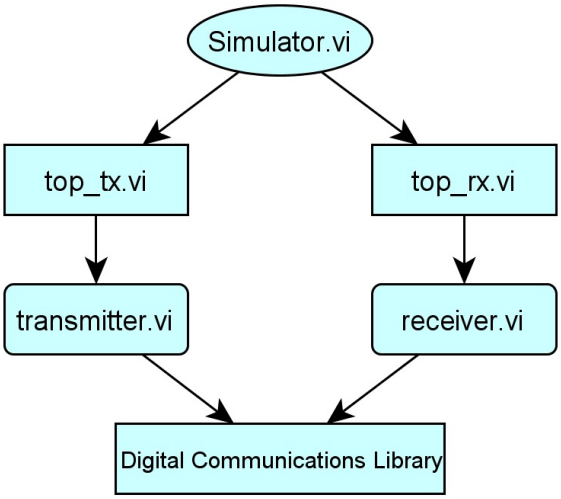

Figure 3: Simulator hierarchy.

Error clusters will allow you to propagate error messages through your code, which will help in debugging your LabVIEW code. If a subVI in your code requires an error cluster as an input, wire *error in* to the appropriate subVI before passing it on to *error out*; otherwise, directly connect these two terminals together.

After you have created your subVIs properly (i.e., wiring all inputs and outputs and editing the icon appropriately), insert them into the simulator provided to you for this lab. To do this, first open up *transmitter.vi* and *receiver.vi* and replace the modulate and decode subVIs with your own (refer to Figures 5 and 6 for where to find these subVIs in the block diagrams). Then you can open up *Simulator.vi* and, using the default parameters, run the simulator to verify that your VIs operate according to the specifications provided in Tables 2 and 3.

Using the simulator and the code you have written, plot average bit-error rate (BER) versus signal-to-noise ratio as in Part 1 of Lab 1, but make sure to obtain data for both BPSK and QPSK modulation[1]. Remember that if signal power is held constant, then decreasing noise power, N_0, is equivalent to increasing SNR [i.e., you can assume unit power, giving SNR $= -N_0$(in dB)]. Also, to observe errors at high SNR (or low N_0) you need to run significantly more iterations in the simulator.

[1] Please ensure that the y-axis is in a logarithmic scale.

Answer the following questions about PSK modulation:

1. PSK is a constant envelope modulation scheme. This means that all symbols have equal energy after modulation. What is the energy of the BPSK and QPSK modulated symbols shown in Tables 1(a) and 1(b) respectively?

2. In *modulate.vi*, what should be the default value for the *Symbol Energy* of the modulated sequence?

3. Consider a uniformly distributed random bit stream $\{b_n\}$ (i.e., $\text{Prob}\{b_n = 1\} = \text{Prob}\{b_n = 0\}$). Suppose $\{b_n\}$ is BPSK modulated such that each symbol has energy E_s. In an AWGN channel, what is the average probability of bit error as a function of SNR when using ML estimation? You may assume that noise is a real-valued Gaussian random variable with variance $\frac{N_0}{2}$, [i.e., $\mathcal{N}(0, \frac{N_0}{2})$].

   HINT: you may leave the answer in terms of the Q-function, which is derived from the $\mathcal{N}(0, 1)$ *distribution.*

Pre-Lab Turn In

1. Submit your implementation of *modulate.vi* and *decode.vi*. Remember, you will be penalized if you do not wire the error cluster inputs and outputs in your code. *Note: If you need to generate any additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.*

2. Turn in a plot which compares the BER curves for BPSK and QPSK modulation. You will need to collect average BER statistics using your code for SNR values of 0 to 10 dB (in 2 dB increments). Remember to use a logarithmic scale for BER and a dB scale for SNR in your plot[2]. You must use your LabVIEW code to generate the data for your plots.

3. Submit your answers to all of the questions in the pre-lab section.

3 Lab Experiment

In this lab you will run your implementation of *modulate.vi* and *decode.vi* over a real wireless link. Remember to use your version of *source.vi* and *error_detect.vi*, which you have already built in Part 1 of Lab 1. Using your

[2] *HINT: If X is a real number, then* $X_{dB} = 10\log X$. *Thus,* $SNR_{dB} = 10\log(\frac{E_S}{N_0})$, *where* E_S *is the average signal power (or symbol energy) and* N_0 *is average noise power.*

code from Part 1 of Lab 1 in conjunction with the new blocks you have created, you will complete the framework for the transmitter and receiver blocks in lab. This framework will be provided in lab. The task of plugging your code into this framework may seem trivial at this point, but as the course progresses you will incrementally re-implement each block in the framework.

Figure 4 describes the hierarchy of files you will be using in lab. Rounded rectangles in the block diagram represent files which you will be able to alter. The files *top_tx.vi* and *top_rx.vi* are the top level of the transmitter and receiver respectively. This level connects the digital communications blocks of the transmitter and receiver (in *transmitter.vi* and *receiver.vi*) with the VIs needed to control the NI RF hardware. You explored the RF hardware and these control VIs in Lab 1, and will need to recall how to modify the parameters/settings for the RF hardware using these VIs.

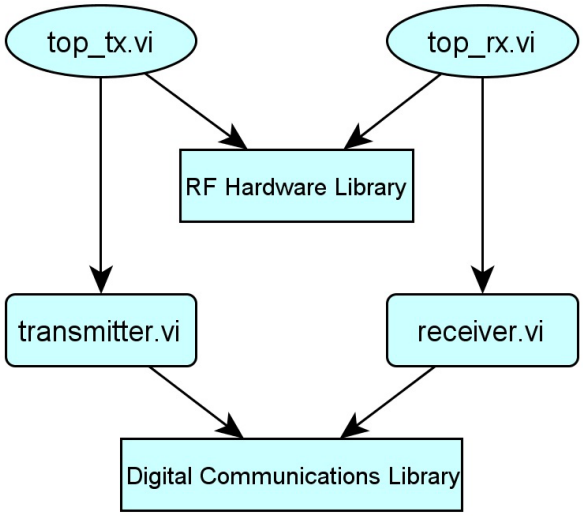

Figure 4: Hierarchy of code dramework used in lab.

3.1 Insert Your Code

The block diagram of *transmitter.vi* is shown in Figure 5. Replace *source.vi* and *modulate.vi* with your code by right-clicking on the appropriate VI and selecting **Replace ⇒Select a VI**.

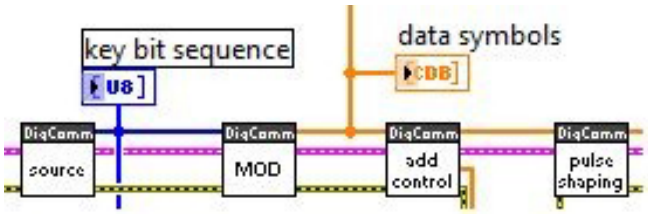

Figure 5: Block diagram of *transmitter.vi*.

Similarly, you will need to replace *decode.vi* and *error_detect.vi* in *receiver.vi* with your code. The block diagram of *receiver.vi* is shown in Figure 6. Add error cluster appropriately.

3.2 Communication Over a Wireless Link

Before you begin testing your code over a real wireless link, set the following baseband parameters on the front panel of *top_tx.vi* and *top_rx.vi*:

- Packet length = 100 bits
- Modulation type = QPSK

Leave all other modulation parameters set to their default value. The following RF parameters have already been set up in the HW parameters tab of *top_tx.vi* and *top_rx.vi* for you:

- Carrier frequency
- TX Gain
- RX Gain
- Trigger Level
- Capture Time

The wireless digital communication system used in lab generates a small burst of data, transmits it over a wireless link, and decodes the received signal. To run the system:

1. Start the top level of your transmitter (i.e., run *top_tx.vi*).
2. Wait until the *Transmitting* indicator is lit.
3. Start the top level of your receiver (i.e., run *top_rx.vi*).

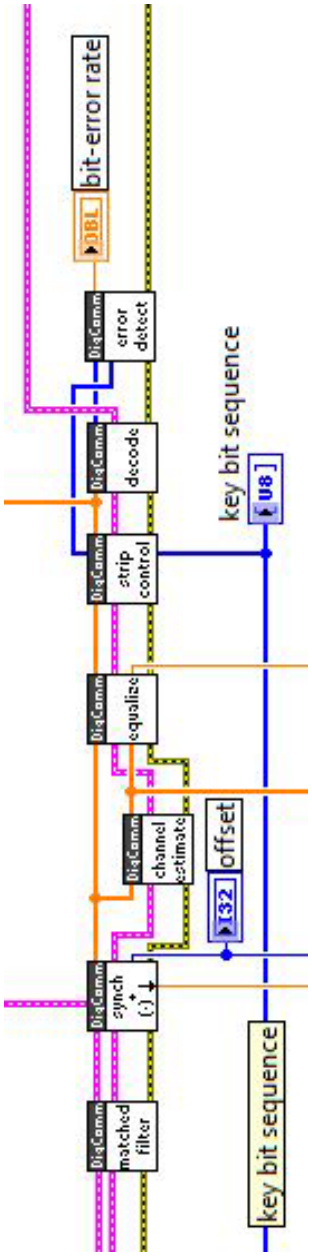

Figure 6: Block diagram of *receiver.vi*.

Open the front panel of *top_rx.vi* to see the constellation of the received signal. Note that a queue has been used to pass the key bit sequence to the receiver for error detection. Also notice, on the front panel of *top_tx.vi* there is a control for the channel model parameters used by *transmitter.vi*. These parameters are used to apply a desired channel model to the transmitted signal. The *real* wireless channel will convolve with this artificially imposed channel model to create a kind of equivalent channel.

Answer the following questions about the digital communication system in lab:

1. What is the name of the queue used to pass the generated bit sequence from *top_tx.vi* to *top_rx.vi* (i.e., the bit sequence needed to perform error detection)?

2. Describe what happens to the received constellation as you increase noise power in the top level of the transmitter, (i.e., *top_tx.vi*).

3. Based on your observation of the received signal constellation, explain which modulation scheme (BPSK or QPSK) performs better (on average) in noisy channels, and why. Assume that both schemes are using the same average symbol energy.

Lab Turn In

Demonstrate to the instructor that your code is working properly. You will need to show the instructor that both BPSK and QPSK modulation work in your system. Please answer the questions above. You will be required to submit these answers in a future lab report (i.e., in Lab 2, Part 2).

Lab 2: Baseband QAM Modem
Part 2: Pulse Shaping and Matched Filtering

Summary

In Part 1 of Lab 2, you implemented the digital modulation and detection parts of a baseband QAM modem. This part of the lab will deal with pulse shaping in digital communication systems. You will implement the pulse shaping and matched filtering blocks of the baseband QAM modem shown in Figure 1.

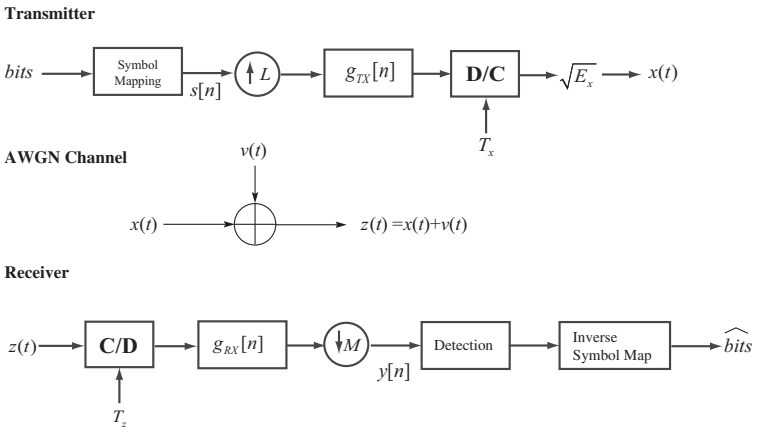

Figure 1: The system under consideration in this lab. The transmitter uses digital pulse-shaping and upsampling to create the transmit waveform. The baseband signal encounters additive white Gaussian noise at the receiver. The receiver uses digital pulse-shaping and downsampling, followed by detection, to find a good guess of the transmitted symbols. Symbol synchronization could be included as well but is not illustrated in the figure.

We begin by discussing the motivation and mathematical model for pulse shaping. You will then build the pulse shaping, up-sampling, and matched filtering blocks of the modem in LabVIEW. After verifying the correctness of your design using a simulator, you will implement your code in lab using the NI-USRP to see how your code works over a real wireless link. The goal of this part of Lab 2 is to understand the motivation for pulse shaping and observe the properties of various pulse shaping filters.

In this lab, you will have to turn in two VIs (*pulse_shaping.vi* and *matched_filtering.vi*), and answer the questions given in Section 2, as part of the pre-lab submission. The lab reports for Parts 1 and 2 of Lab 2 have to be submitted as a single lab report at the end of this lab.

1 Pulse Shaping and Matched Filtering

The objective of the course is to construct the digital signal processing blocks necessary to operate a wireless digital communication link. A digital signal (from the digital source, constructed in Part 1 of Lab 1) can not be transmitted over a communication channel. The digital signal needs to be mapped to an analog waveform that can easily be transmitted over the channel.

The model for transmission and reception considered in Lab 2 is illustrated in Figure 1. The transmitter creates a complex pulse amplitude modulated signal of the following form:

$$x(t) \quad = \quad \sqrt{E_x} \sum_{n=-\infty}^{\infty} s[n] g_{\text{tx}}(t - nT). \tag{1}$$

Conceptually, the mapping of bits to waveforms can be viewed in two steps.

1. **Source bits to symbols**—This involves mapping the bits emitted by the (digital) source to (possibly complex) symbols, s on a constellation, $\mathcal{C}$ determined by the modulation scheme. This was accomplished in Part 1 of Lab 2.

2. **Symbols to pulses**—This involves creating an analog pulse train to be transmitted over the wireless channel, using the complex symbols generated in the first step. This is done using a pulse-shaping filter $g_{\text{tx}}(t)$ at the transmit side.

The receiver processing depends on the type of distortion in the communication channel. Assuming an additive white Gaussian noise (AWGN) channel, the receiver performs analogous operations as at the transmitter.

1. **Pulses to symbols**—In this step, the symbols are recovered from the received pulses using a matched filter at the receive side.

2. **Symbols to bits**—This involves mapping the symbols received to bits (depending on the modulation scheme chosen), using some form of a detection technique. This was accomplished in Part 1 of Lab 2 using the maximum likelihood detection algorithm.

To implement the maximum likelihood detector as described in Part 1 of Lab 2 using single-shot detection, the pulse shaping filters $g_{\text{tx}}(t)$ and $g_{\text{rx}}(t)$ have to satisfy some properties.

- The transmit pulse shape is normalized such that $\int |g_{\text{tx}}(t)|^2 dt = 1$. This is primarily for convenience in our notation to avoid additional scaling factors.
- The receive filter $g_{\text{rx}}(t) = g_{\text{tx}}^*(-t)$, which simplifies to $g_{\text{rx}}(t) = g_{\text{tx}}^*(-t)$ since the pulse-shaping function is usually real. This means that the receive filter is a matched filter. Effectively the filtering by $g_{\text{rx}}(t)$ becomes a correlation with $g_{\text{tx}}(t)$ thanks to the relationship between convolution and correlation. The choice of a matched filter maximizes the received signal-to-noise ratio and thus gives the maximum likelihood detector its best performance.
- The composite filter $g(t) = \int g_{\text{rx}}(\tau) g_{\text{tx}}(t - \tau) dt$ is a Nyquist pulse shape. This means that it satisfies $g(nT) = \delta[n]$. The Nyquist pulse-shape property eliminates intersymbol interference, allowing the detector to take a very simple form as described in Part 1 of Lab 2.

Defining the bandwidth of the transmitted signal $x(t)$ requires some mathematical subtleties. From the perspective of measuring the power spectrum of a signal, the following definition of power spectrum for the transmitted signal is sufficient:

$$P_x(f) \equiv E_x \, |G_{\text{tx}}(f)|^2 \tag{2}$$

where it is assumed that the transmit constellation is normalized to unit energy. It is clear that $|G_{\text{tx}}(f)|^2$ determines the bandwidth of $x(t)$, where $G_{\text{tx}}(f)$ is the frequency response of the pulse-shape $g_{\text{tx}}(t)$. By normalizing the transmit pulse shape, $\int_f P_x(f) df = E_x$ where E_x is known as the symbol energy and E_x/T as the power.

A simple choice for $g_{\text{tx}}(t)$ is the rectangular pulse-shape. Unfortunately, such a pulse-shape would have poor spectral properties, since the Fourier transform of a rectangular function is a sinc function. This means that a lot of bandwidth would be required to transmit the signal.

Another choice for $g_{\text{tx}}(t)$ is the sinc function. A sinc pulse has ideal spectral properties. For example, if $g_{\text{tx}}(t) = \frac{\sin(\pi t/T)}{\pi t/T}$ then $G_{\text{tx}}(f) = T$ for $f \in [-1/2T, 1/2T]$ giving an absolute bandwidth of $1/2T$. This is the smallest bandwidth a complex pulse amplitude modulated signal could have with symbol rate $1/T$. Unfortunately the sinc pulse shaping filter has a number of problems in practice. Ideal implementations of $g(t)$ do not exist in analog. Digital implementations require truncating the pulse shape, which is a problem since it decays in time with $1/t$. Further the sinc function is sensitive

to sampling errors (not sampling at exactly the right point). For these reasons it is of interest to consider pulse shapes that have excess bandwidth, which means the double sided bandwidth of the pulse-shape is greater than $1/T$.

1.1 A Common Pulse Shape

The most common Nyquist pulse, besides the sinc pulse, is the raised cosine. The raised cosine pulse-shape has Fourier spectrum

$$G_{\text{rc}}(f) = \begin{cases} T, & 0 \le |f| \le (1-\alpha)/2T \\ \frac{T}{2}\left[1+\cos\frac{\pi T}{\alpha}\left(|f|-\frac{1-\alpha}{2T}\right)\right], & \frac{1-\alpha}{2T} \le \frac{1+\alpha}{2T} \\ 0, & |f| > \frac{1+\alpha}{2T} \end{cases}$$

and transform

$$g_{\text{rc}} = \frac{\sin \pi t/T}{\pi t/T}\frac{\cos(\pi\alpha t)}{1-4\alpha^2 t^2/T^2}.$$

The parameter α is the rolloff factor, $0 \le \alpha \le 1$ (sometimes β is used instead of α) and is often expressed as percentage excess bandwidth. In other words, 50% excess bandwidth would correspond to $\alpha = 0.5$. At large values of α the tails of the time-domain pulse decay quickly, but the spectrum roll off is smoother (i.e., it requires more excess bandwidth). Reducing the time-domain tails of the pulse shape reduces the amount of ISI introduced by sampling error. Thus rolloff factor α can be used to tradeoff excess bandwidth for resistance to ISI caused by sampling errors. The raised cosine pulse is illustrated in Figure 2.

The transmitter does not use the raised cosine pulse directly. The reason is that the raised cosine pulse shape corresponds to $g(t)$ while we use $g_{\text{tx}}(t)$ at the transmitter and $g_{\text{rx}}(t)$ at the receiver. Recall the $g(t) = \int g_{\text{rx}}(\tau)g_{\text{tx}}(t-\tau)d\tau$ and that $g_{\text{rx}}(t) = g_{\text{tx}}(-t)$. In the frequency domain this means that $G(f) = G_{\text{tx}}(f)G_{\text{tx}}^*(f)$. Consequently we choose $g_{\text{tx}}(t)$ to be a "square root" of the raised cosine. Such a pulse shape is known as a square root raised cosine, or a root raised cosine and is written

$$g_{\text{sqrc}}(t) = \frac{4\alpha}{\pi\sqrt{T}}\frac{\cos\left[(1+\alpha)\pi t/T\right] + \frac{\sin[(1-\alpha)\pi t/T]}{4\alpha t/T}}{1-(4\alpha t/T)^2}.$$

Of particular interest note that $g_{\text{sqrc}}(t)$ is even, thus if $g_{\text{tx}}(t) = g_{\text{sqrc}}(t)$ then $g_{\text{rx}}(t) = g_{\text{tx}}(-t) = g_{\text{tx}}(t)$ and the transmit pulse shape and receive pulse shape are identical. The square root raised cosine is its own matched filter!

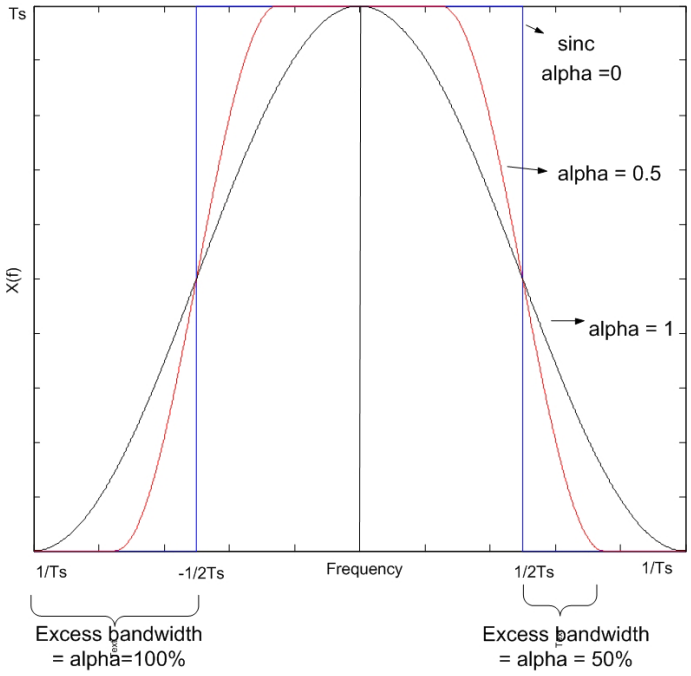

Figure 2: Spectral response of the raised cosine pulse shaping function $g(t)$.

1.2 Practical Implementation of Pulse Shaping

Recall that the Nyquist theorem states it is sufficient to sample a given signal at a sampling frequency is at least twice the largest frequency in the signal, to be able to reconstruct the signal effectively. Sampling at a rate higher than what is required by the Nyquist theorem is known as oversampling. Oversampling at the analog front ends of a digital communication system (DAC and ADC) will make the design of all the filters much simpler. The increased bandwidth, due to a larger sampling frequency, will relax the sharp cutoff requirements on the filters needed to remove aliasing effects. Oversampling in digital systems is also important for efficient synchronization.

The upsampling and downsampling processes are critical in order to reap the benefits of oversampling. Upsampling a sequence $\{s[n]\}$ by a factor L is equivalent to inserting $L-1$ zeros between every sample of the sequence. Downsampling a sequence $\{y[n]\}$ by a factor M is equivalent to throwing away $M-1$ out of every M symbols (i.e., $y_{\text{downsamp}}[k] = y[kM]$). Upsampling and downsampling are used to facilitate the digital implementation of $g_{\text{tx}}(t)$ and $g_{\text{rx}}(t)$.

Simple explanations of transmit and receive pulse shaping using the concepts of upsampling and downsampling are provided in this lab. More efficient implementation computationally is possible using filter banks but that is not required to complete the assignment.

Transmit Pulse Shaping

The challenge with implementing transmit pulse shaping is that the symbol rate may be less than the sample rate required for pulse shapes with excess bandwidth. For example, for a square root raised cosine pulse shape, the Nyquist rate is $(1+\alpha)/T$ while the symbol rate is only $1/T$.

Let T_x be some sampling period such that $1/T_x$ is greater than twice the maximum frequency of $g_{\text{tx}}(t)$. For simplicity we take $T_x = T/L$. Other choices of T_x would require a resampling operation. The continuous-time complex baseband signal is

$$x(t) = \sqrt{E_x} \sum_{n=-\infty}^{\infty} s[n] g_{\text{tx}}(t - nT). \tag{3}$$

Since $x(t)$ is bandlimited by virtue of the bandlimited pulse shape $g_{\text{tx}}(t)$, there exists a sequence $\{c[n]\}$ such that

$$x(t) = \sqrt{E_x} \sum_{n=-\infty}^{\infty} c[n] \frac{\sin((t - nT_x)/T_x)}{(t - nT_x)/T_x},$$

where $c[n] = x(nT_x)$. Substituting Eq. (3) gives

$$\begin{aligned} c[n] &= \sum_{m=-\infty}^{\infty} s[m] g_{\text{tx}}(nT_x - mT) \\ &= \sum_{m=-\infty}^{\infty} s[m] g_{\text{tx}}[n - mL], \end{aligned} \tag{4}$$

where $g_{\text{tx}}[n] = g_{\text{tx}}(nT_x)$. Using multi-rate signal processing identities, it can be shown that Eq. (4) is the result of upsampling $s[m]$ by L, then filtering with $g_{\text{tx}}[n]$.

Receiver Pulse Shaping

It is easier to justify the digital implementation of receiver pulse shaping. Let $z(t)$ denote the complex baseband input to the continuous-to-discrete converter. Assume that $z(t)$ has already been bandlimited by the RF in the

analog front end. Let $T_z = T/M$ for some integer M such that $1/T_z$ is greater than the Nyquist rate of the signal. This is known as *oversampling*. Let $z[n] := z(nT_z)$. Using results on filtering bandlimited signals, it can be shown that

$$y[n] = \int g_{\text{rx}}(\tau) z(nT - \tau) \quad = \quad T_z \sum_{m=-\infty}^{\infty} z[m] g_{\text{rx}}((nM - m)T_z) \quad (5)$$

$$= \quad \sum_{m=-\infty}^{\infty} z[m] g_{\text{rx}}[nM - m], \quad (6)$$

where $g_{\text{rx}}[n] = T_z g_{\text{rx}}(nT_z)$. Using multi-rate signal processing identities, it can be shown that Eq. (6) is the result of downsampling the filtered oversampled signal $z[n]$ by $g_{\text{rx}}[n]$.

1.3 Pulse Shaping in LabVIEW

LabVIEW provides many useful digital signal processing tools. In this lab, you will use *MT Generate Filter Coefficients.vi*, a Modulation Toolkit VI found in the Modulation palette (**Function⇒Addons⇒Modulation⇒Digital⇒Utilities⇒MT Generate Filter Coefficients.vi**). This Modulation Toolkit VI allows you to generate a pulse shaping filter (e.g., root raised cosine) and its associated matched filter based on a set of input parameters. You will need to use this VI to build the pulse shaping and matched filtering blocks of Figure 1. The inputs to this VI [*modulation type*, *pulse shaping filter*, *filter parameter* and *filter length (symbols)*] can be obtained from the *pulse shaping parameters* available in the *modulation parameters* cluster.

As a note about code style in LabVIEW, when building a VI requiring many inputs/outputs (i.e., about five or more of each), you should bundle related inputs/outputs into clusters. Clusters, found in the Cluster palette, allow you to group together terminals of different datatypes. Please refer to [12] for more information on how to build a cluster or access the elements of a cluster. For an example of a cluster you will be using in the lab, please look at the *pulse shaping parameters* terminal, which can be accessed from the front panel of *top_tx.vi*.

2 Pre-Lab

To prepare for this part of the lab you will be required to build the pulse shaping and matched filtering blocks of Figure 1. Tables 1 and 2 describe the details of the VIs you must implement. You will plug your code into *receiver.vi* and *transmitter.vi* and run the simulator provided to you in order

Table 1: Description of *pulse_shaping.vi*

pulse_shaping.vi - oversample input and apply transmit pulse shaping filter			
INPUTS	*input*	array of CDB (complex doubles)	input symbol sequence
OUTPUTS	*output*	array of CDB	oversampled input convolved with pulse shaping filter

Table 2: Description of *matched_filtering.vi*

matched_filtering.vi - apply matched filter corresponding to pulse shaping parameters			
INPUTS	*input complex waveform*	IQ Waveform cluster[1]	received sequence
OUTPUTS	*output complex waveform*	IQ Waveform cluster	received sequence convolved with matched filter

to verify functionality and correctness. Please name your VIs as per the convention *yourname_VIname.vi*.

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break.

Figure 3 shows the dependencies between the files that make up the simulator. The top level of this simulator, *simulator.vi*, connects *top_tx.vi* to

[1] IQ Waveform clusters contain three elements: *t0*, *dt*, and *Y*. Please recall from Part 2 of Lab 1 what these elements are and what their data types are.

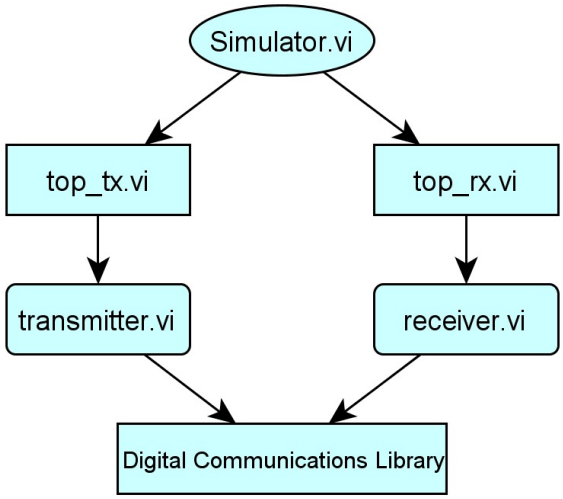

Figure 3: Hierarchy of code framework for new simulator.

top_rx.vi and provides each with the appropriate inputs. The parts of the simulator you will be modifying are located in *transmitter.vi* and *receiver.vi* shown in Figures 4 and 5 respectively.

You will be putting your VIs into transmitter.vi and receiver.vi, replacing the locked versions that are already there. After doing this, you will then open up simulator.vi, that you will use to confirm your VIs operate correctly before implementing them on the NI-USRP.

Notice that *pulse_shaping.vi* and *matched_filtering.vi* do not take any parameters as inputs. All of the pulse shaping and oversampling parameters you need to use for these VIs can be accessed from the *modulation parameters in* cluster. After building these VIs, replace the existing code in the simulator with your code. Replace a subVI in the transmitter or receiver with your code

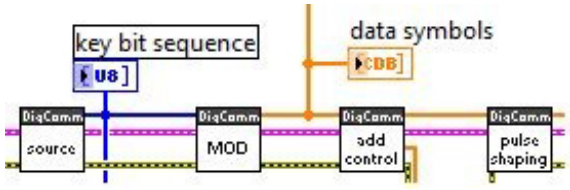

Figure 4: Block diagram of *transmitter.vi*.

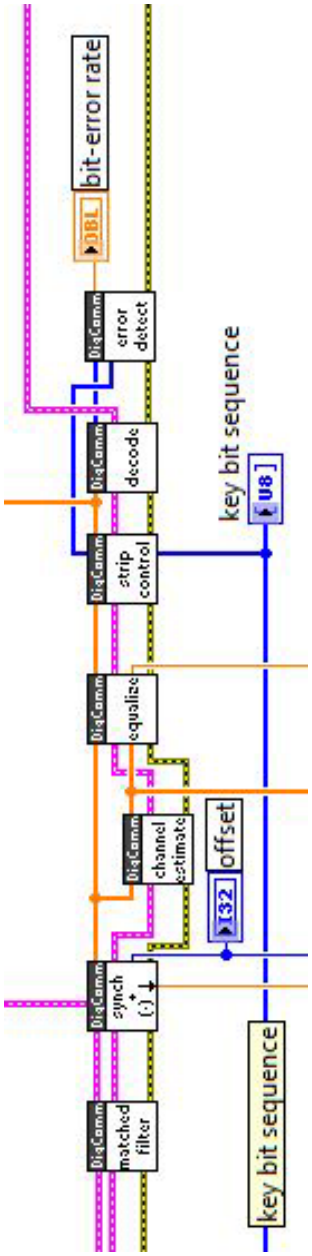

Figure 5: Block diagram of *receiver.vi*.

by right-clicking on the appropriate VI and selecting **Replace⇒Select a VI**. Remember to wire error clusters where appropriate.

As described in the previous section, use *MT Generate Filter Coefficients.vi* from the Modulation Toolkit to generate your pulse shaping and matched filters. After generating each filter, you can apply the filter by convolving it with the complex array component of the appropriate input waveform. Do not use a Modulation Toolkit VI to apply either the pulse shaping filter or matched filter.

In LabVIEW, *Convolution.vi*, which is located in the Signal Processing tool palette (**Functions⇒Signal Processing⇒Signal Operation⇒ Convolution.vi**), supports complex convolution. You can use this VI when creating your subVIs for this lab and others.

In summary:

- Use *MT Generate Filter Coefficients.vi* to generate the pulse shaping filter and its associated matched filters. Do not use Modulation Toolkit VIs to apply these filters. Instead, use convolution, and remember to make any necessary changes needed for complex convolution.
- The *modulation parameters* cluster contains many of the parameters needed for pulse shaping, including *pulse shaping parameters*, *TX oversample factor*, and *RX oversample factor*.
- After building *pulse_shaping.vi* and *matched_filtering.vi*, insert them into the simulator provided to you to verify functionality and correctness.

Using an Eye Diagram as an Analytical Tool

In this lab you will use an eye diagram to make some qualitative statements about the received signal. An eye diagram is constructed by taking a time domain signal and overlapping traces of a certain number of symbol times. For example, if $T_s = 10$ μsec is the symbol time for a given waveform, an eye diagram for an eye length of two would be generated by overlaying traces of duration $2T_s$ starting at an ideal sampling point. In the absence of noise, these overlapped signals might look something like Figure 6.

The eye diagram of Figure 6 shows the overlapped traces of a BPSK modulated bit stream using a sinc pulse for an eye length of two. This eye diagram can give us some qualitative insight about our received signal. For example, the opening of the eye indicates the period of time over which we can successfully sample the waveform (i.e., without suffering from ISI). Along with indicating the amount of ISI in the system, the distortion of the signal in the diagram also indicates various qualitative parameters, such as

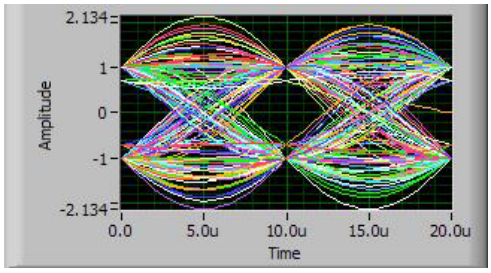

Figure 6: Eye diagram for sinc pulse shape in the absence of noise.

the variation of zero crossings or distortion related to SNR. Please refer to [6] and [1] for more detailed discussions about the use of the eye diagram as a tool for qualitative analysis in digital communications.

Questions

Answer the following questions, and submit the answers as part of your pre-lab.

1. *MT Generate Filter Coefficients.vi* requires an input for the number of *pulse shaping samples per symbol*. This tells the VI the over-sample factor being used to design the filter. When using this VI in *pulse_shaping.vi*, where do you obtain this parameter from? Give any relevant cluster names and/or variable names if appropriate.

2. According to [1], impairments visible in the eye diagram can occur at many places along the communication path (i.e., from source to sink). What are some of the places where impairments visible in the eye diagram might occur (name at least three)?

3. When using inconsistent sampling rates at the transmitter and receiver, it is important to remember that the symbol rate ($1/T_s$) must remain constant.

   (a) In terms of the sampling rates at the transmitter and receiver ($1/T_M$ and $1/T_N$ respectively), what is the relation between the oversample factor at the transmitter (M) and receiver (N)?

   (b) What can you say about the type of number the ratio of $\frac{T_M}{T_N}$ (i.e., the ratio of the sampling rate of the receiver to that of the transmitter) must be?

   Hint: Note that M and N must be positive integers.

Pre-Lab Turn In

1. Submit your implementation of *pulse_shaping.vi* and *matched_filtering.vi*. Remember, you will be penalized if you do not wire the error cluster inputs and outputs in your code.

   Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section.

3 Lab Experiment

In this lab you will run your implementation of *pulse_shaping.vi* and *matched_filtering.vi* over a real wireless link. Using your code from the prior labs in conjunction with the new blocks you have created, you will complete the framework for the transmitter and receiver blocks in this lab. This framework will be provided to you as in the Part 1 of Lab 2.

Figure 7 describes the hierarchy of files you will use in this lab. Rounded rectangles in the block diagram represent files you will be able to alter. The files *top_tx.vi* and *top_rx.vi* are the top level of the transmitter and receiver respectively. This level connects the digital communications blocks of the transmitter and receiver (in *transmitter.vi* and *receiver.vi*) with the VIs needed to control the NI-USRP. You explored the USRP and these control VIs in Lab 1, and will need to recall how to modify the parameters/settings for the USRP using these RF hardware VIs.

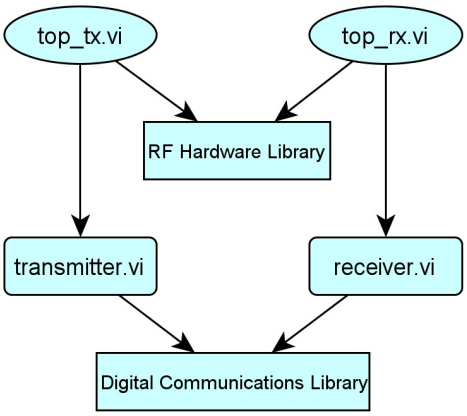

Figure 7: Hierarchy of code framework used in lab.

3.1 Communication Over a Wireless Link

Before you begin testing your code over a real wireless link, set the following baseband parameters on the front panels of *top_tx.vi* and *top_rx.vi*:

- Packet length = 200 bits
- Modulation type = QPSK
- Pulse shaping filter = Root Raised Cosine
- Filter parameter = 0.5
- Filter length = 8

Also make sure that the following RF parameter is set up appropriately (using the HW parameters accessible on the front panel of *top_rx.vi*):

- Capture time = 2 msec

Let us digress now to clarify the operation of *top_rx.vi*, the top level of the receiver. The receiver has two states: (1) *listen* and (2) *receive*. Figure 8 shows a finite state diagram of the operation of the receiver. The *RXRF_recv.vi* features a digital triggering subVI. When this feature is enabled, the digitizer will continually "listen" to the received signal until the energy of the incoming signal crosses a predefined trigger level. Once this trigger occurs, the NI-USRP captures a window of samples, which it then passes to *receiver.vi*. After processing the captured signal, the receiver returns to the listening state. The receiver will periodically timeout while in the *listen* state to update RF configuration parameters.

Now you are ready to test your code over a real wireless link. The wireless digital communication system used in the lab generates a small burst of data,

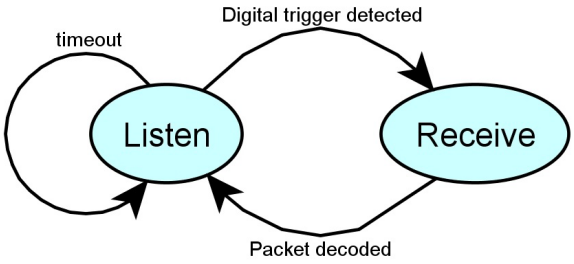

Figure 8: State diagram for receiver implemented by *top_rx.vi*.

transmits it over a wireless link, and decodes the received signal. To run the system:

1. Start the top level of your transmitter (i.e., run *top_tx.vi*).
2. Wait until the *Transmitting* indicator is lit.
3. Start the top level of your receiver (i.e., run *top_rx.vi*).
4. Open the front panel of *top_rx.vi* to see the constellation and eye diagram of the received signal.

Note that a queue has been used to pass the key bit sequence to the receiver for error detection. Also notice on the front panel of *top_tx.vi* there is a control for the channel model parameters used by *transmitter.vi*. These parameters are used to apply a desired channel model to the transmitted signal. The *real* wireless channel will convolve with this artificially imposed channel model to create a kind of equivalent channel.

After you have verified the correct operation of your code using the parameters above, vary the input parameters to the system and observe how these changes affect the eye diagram of the received signal. Some of the parameters you might vary include noise power, pulse shape, filter parameter, or channel model (e.g., you may enter your own channel impulse response using the ISI channel model).

3.2 Questions

Answer the following questions about the digital communication system:

1. Using the *pulse shaping parameters* control on the front panels of *top_tx.vi* and *top_rx.vi*, vary the filter parameter (i.e., the rolloff factor) of a root raised cosine pulse from 0 to 1 in increments of 0.20. Observe how the eye diagram (on the front panel of *top_rx.vi*) changes as you vary this excess bandwidth parameter.
   (a) Describe what happens to the opening of the eye as you increase the excess bandwidth parameter from 0 to 1.
   (b) Explain what impairment is being added or removed from the received signal as you vary this parameter.
2. Based on what you have learned from [1], and from varying the pulse shaping parameters, what is the relation between the excess bandwidth parameter of the root raised cosine pulse and inter-symbol interference? Does ISI increase or decrease with increasing α and why.

Lab Turn In

Demonstrate that your code is working properly. You will need to show that your pulse shaping VIs can work with a variety of pulse shaping parameters. Also answer the questions above. You will be required to submit these answers in your lab report.

DIRECTIONS FOR LAB SUBMISSION

Your lab report should accomplish the following tasks.

1. Answer all questions from Parts 1 and 2 of Lab 2 (not including pre-lab questions). Note that this also includes the questions that you answered during the lab session.

2. Discuss any problems you might have encountered in either Part 1 or 2 of Lab 2 and how you overcame these obstacles.

3. Describe how the code you have developed in lab for *modulate.vi* and *decode.vi* might be extended or generalized to be used with an arbitrary constellation (i.e., if you were given some arbitrary constellation $\mathcal{C}$ as a 1-D array of complex doubles in LabVIEW, explain how you would modify your code for *modulate.vi* and *decode.vi* to use this new constellation). Please only explain details of your code that pertain to your implementation of algorithms for symbol mapping, energy normalization, and maximum-likelihood detection. You do not have to construct the VI, only an explanation is needed.

4. The background section of this lab stated that two main goals for pulse shaping are:

   - To reduce the energy of spectral sidelobes (i.e., excess bandwidth requirements).
   - To reduce the time-domain pulse tails which can lead to inter-symbol interference.

   Are these conflicting or complimentary goals and why?

Lab 3: Synchronization: Symbol Timing Recovery in Narrowband Channels

Summary

In this lab you will consider the problem of symbol timing recovery also known as symbol synchronization. Timing recovery is one of several synchronization tasks; others will be considered in future labs.

The wireless communication channel is not well modeled by simple additive white Gaussian noise. A more realistic channel model also includes attenuation, phase shifts, and propagation delays. Perhaps the simplest channel model is known as the frequency flat channel. The frequency flat channel creates the received signal

$$z(t) \quad = \quad \alpha e^{j\phi} x(t - \tau_d) + v(t), \tag{1}$$

where α is an attenuation, ϕ is a phase shift, and τ_d is the delay.

The objective of this lab is to correct for the delay caused by τ_d in discrete-time. The approach will be to determine an amount of delay $\hat{k}$ and then to delay the filtered received signal by $\hat{k}$ prior to downsampling. This will modified the receiver processing as illustrated in Figure 1.

Two algorithms will be implemented for symbol synchronization in this lab: the maximum energy method and the Early Late gate algorithm. The maximum energy method attempts to find the sample point that maximizes the average received energy. The early–late gate algorithm implements a discrete-time version of a continuous-time optimization to maximize a certain

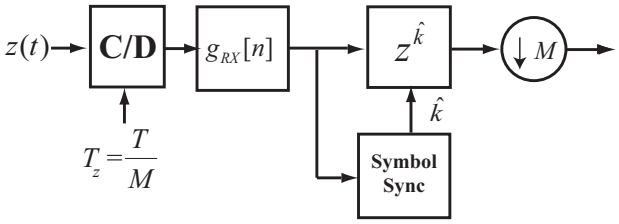

Figure 1: Receiver with symbol synchronization after the digital matched filtering.

cost function. For the best performance, both algorithms require that there is a lot of oversampling (i.e., M is large).

You will build two VIs, one for each of the two algorithms. After verifying the correctness of your design using a simulator, you will implement your code in lab using the NI-USRP hardware to see how your code works over a real wireless link. The goal of this lab is to understand the need for symbol timing recovery and the performance of two common symbol timing recovery algorithms.

In this lab, you will have to turn in two VIs (*align_MaxEnergy.vi* and *align_ELgate.vi*) and submit the answers to the questions in Section 2, along with the plot depicting timing error statistics, as part of the pre-lab submission.

1 Background

This section provides some background on the timing recovery problem and reviews the two algorithms that will be implemented in this lab. Remember that you will be implementing these synchronization methods using a purely digital approach.

1.1 Introduction to Symbol Timing Recovery

In the presence of frequency-flat fading as described in Eq. (1), the input to the receiver has the form

$$z(t) = \alpha e^{j\phi}\sqrt{E_x}\sum_m s[m]g_{\text{tx}}(t - mT - \tau_d) + v(t). \tag{2}$$

Now let $g(t)$ be the convolution of $g_{\text{tx}}(t)$ and $g_{\text{rx}}(t)$. With the model in Eq. (2) after matched filtering and sampling, the received signal is

$$y[n] = \sqrt{E_x}\alpha e^{j\phi}\sum_m s[m]g((n-m)T - \tau_d) + v[m].$$

Several different impairments are possible depending on the value of the τ_d. Unless otherwise specified, Nyquist pulse shapes are assumed.

First consider the case where τ_d is a fraction of a symbol period thus $0 < \tau_d < T$. This can model the effect of sample timing error, or not sampling at the right point. Under this assumption

$$y[n] = \underbrace{\sqrt{E_x}\alpha e^{j\phi}s[n]g(\tau_d)}_{\text{desired}} + \underbrace{\sqrt{E_x}\alpha e^{j\phi}\sum_{m\neq n} s[m]g((n-m)T - \tau_d)}_{\text{ISI}} + \underbrace{v[n]}_{\text{noise}}.$$

Intersymbol interference is created when the Nyquist pulse shape is not sampled exactly at nT. You will need *symbol synchronization* or *equalization* to mitigate the effect of sample timing error.

Second, suppose that $\tau_d = dT$ for some integer d. This is the case where symbol timing has been resolved but an unknown propagation delay, which is a multiple of the symbol period, remains. Under this assumption

$$\begin{aligned} y[n] &= \sqrt{E_x}\alpha e^{j\phi} \sum_m s[m]g((n-m)T - \tau_d) + v[m] \\ &= \sqrt{E_x}\alpha e^{j\phi} \sum_m s[m]g((n-m)T - dT) + v[m] \\ &= \sqrt{E_x}\alpha e^{j\phi} s[n-d] + v[m]. \end{aligned}$$

Essentially integer offsets create a mismatch between the indices of the transmitted and receive symbols. You will need *frame synchronization* to find the beginning of the data.

Third, suppose that the time error has been removed from the channel; equivalently $\tau_d = 0$. Then

$$y[n] = \sqrt{E_x}\alpha e^{j\phi} s[n] + v[m].$$

Sampling and removing delay leaves distortion due to α and ϕ, which are unknown to the receiver. The amplitude and phase shift will be estimated and equalized in Lab 4 (the focus of this lab is on the unknown delay τ_d).

1.2 The Maximum Output Energy Solution

Let $y(t)$ be the matched filtered version of $z(t)$ in continuous-time. Given an advance τ, notice that the output energy function defined as

$$\begin{aligned} J(\tau) = \mathbb{E}\,|y\,(nT+\tau)|^2 &= E_x \sum_m |g\,(mT + \tau - \tau_d)\,|^2 + \sigma_v^2 \\ &\leq E_x |g(0)|^2 + \sigma_v^2. \end{aligned}$$

The maximum of the function $J(\tau)$ occurs when $\tau - \tau_d$ is an integer multiple of the symbol rate. This value of τ is known as the maximum output energy solution. The resulting value of τ by that maximizes $J(\tau)$ is known as the maximum output energy solution. In this lab we implement two algorithms for finding a maximum output energy solution in discrete-time. Other criteria for performing symbol synchronization are possible, for example maximum likelihood solutions. The approach described here is robust to additive noise, certain kinds of fading channels, and small carrier frequency offsets.

You will implement a digital version of the maximization of $J(\tau)$. This means that you will perform the estimation and correction after the digital matched filter at the receiver, prior to downsampling.

1.3 Direct Maximization of the Output Energy in Discrete Time

Let

$$r[n] = \sum_m z[m] g_{\text{tx}}[n-m]$$

be the output of the matched receiver filter. The discrete-time output energy can be calculated as

$$J[k] = \mathbf{E}\left|r\left(nMT+k\right)\right|^2,$$

where k is the sample offset between $0, 1, \ldots, M-1$ corresponding to an estimate of $\hat{\tau} = kT/M$. The expectation can be replaced with a time average over P symbols to create the function

$$J_{\text{approx}}[k] = \frac{1}{P}\sum_{p=0}^{P-1}\left|r\left(pMT+k\right)\right|^2. \tag{3}$$

The direct maximum output energy solution is given by $\hat{k} = \max J_{\text{approx}}[k]$. Because the receiver is only performing symbol synchronization, it is sufficient to evaluate $J_{\text{approx}}[k]$ for values $k = 0, 1, \ldots, M-1$. The resulting value of $\hat{k}$ would be implemented in the advance operation in Figure 1 given in the Z-domain as $z^{\hat{k}}$. Instead of an advance, a delay could be used instead with value $z^{M-\hat{k}}$. Larger values of P generally give better performance.

1.4 Indirect Maximization of the Output Energy in Discrete Time

An alternative approach, useful for adaptive implementations, involves differentiating the cost function and discretizing the result. The approach here follows the description from [6]. Differentiating the expectation and assuming they can be interchanged

$$\begin{aligned}\frac{d}{d\tau}J_{opt}(\tau) &\simeq E\left\{\frac{d}{d\tau}\left|y\left(nT_s+\tau\right)\right|^2\right\}\\ &= E\left\{y(nT+\tau)\frac{\delta}{\delta\tau}y^*(nT+\tau)\right\}+E\left\{y^*(nT+\tau)\frac{\delta}{\delta\tau}y(nT+\tau)\right\}\\ &= 2\text{Re}\left[E\left\{y(nT+\tau)\frac{\delta}{\delta\tau}y^*(nT+\tau)\right\}\right].\end{aligned}$$

Using a first-order difference approximation

$$\frac{\delta}{\delta\tau}y^*(nT+\tau) \simeq y^*\left(nT_s+\tau+\delta\right) - y^*\left(nT_s+\tau-\delta\right).$$

Now as before replacing the expectation with a time average

$$\frac{d}{d\tau}J_{opt}(\tau) \simeq \frac{1}{P}\sum_{n=1}^{P-1} 2\mathrm{Re}\left\{y\left(nT_s+\tau\right)\left(y^*\left(nT_s+\tau+\delta\right)-y^*\left(nT_s+\tau-\delta\right)\right)\right\}.$$

Choose δ to be a multiple of T/M. Using $r[n]$ as defined before let

$$J_\delta[k] = \sum_{n=0}^{P-1} 2\mathrm{Re}\left\{r\left[nP+k\right]\left(r^*\left[nP+k+\delta\right]-r^*\left[nP+k-\delta\right]\right)\right\}. \quad (4)$$

What we call the indirect maximum output energy solution is given by $\hat{k} = \max_{k=0,1,\ldots,M-1} J_\delta[k]$. A typical value of δ is 1.

2 Pre-Lab

Please answer the following questions about symbol timing recovery:

1. Show that in the absence of noise, α and ϕ in Eq. (2) do not have any impact on the maximum output energy solution.

2. What are the two critical assumptions used to formulate the indirect maximization of the output energy? Consider how the presence of the flat fading channel AWGN can impact this method. Specifically, using at least one of the critical assumptions, explain how you might mitigate the impact of these impairments by suitable selection of parameters.

3. After downsampling a sequence originally sampled at rate $1/T_z$ by a factor M, what is the sample period of the resulting signal?

Programming

In this pre-lab you will generate LabVIEW code to estimate the direct maximization of the output energy in discrete time and the early–late gate algorithm. Tables 1 and 2 describe the details of the VIs you must implement. Please name your VIs as per the convention *yourname_VIname.vi*.

Table 1: Description of *align_MaxEnergy.vi*

align_MaxEnergy.vi - compute sampling offset by maximizing $J_{\text{approx}}[k]$ and correct for offset			
INPUTS	*input complex waveform*	IQ Waveform cluster	received sequence after matched filtering
OUTPUTS	*output complex waveform*	IQ Waveform cluster	received sequence after symbol timing recovery
	offset	I32 (integer)	computed offset

Table 2: Description of *align_ELgate.vi*

align_ELgate.vi - estimate timing offset by minimizing $J_{\delta}[k]$ and correct for offset			
INPUTS	*input complex waveform*	IQ Waveform cluster	received sequence after matched filtering
OUTPUTS	*output complex waveform*	IQ Waveform cluster	received sequence after symbol timing recovery
	offset	I32 (integer)	computed offset

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break.

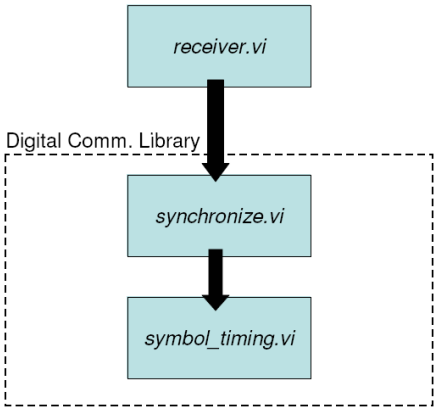

Figure 2: Hierarchy of code needed for symbol timing recovery in lab.

You will plug your code into the baseband simulator provided to you in order to verify functionality and correctness. You will not be replacing one of the main blocks inside *receiver.vi* as you have done in previous labs. Instead, you will need to insert your code into *symbol_timing.vi*, a subVI in the digital communications library built for this course. Figure 2 describes the hierarchy of the files needed for symbol timing recovery in this lab.

Figure 2 shows the block diagram of *synchronize.vi*, where you can find *symbol_timing.vi*. Notice there are a number of synchronization methods available in this VI, however you will only be dealing with *"Timing Estimation"* (the option for performing symbol timing recovery independent of frame synchronization or channel estimation).

After replacing *align_ELgate.vi* and *align_MaxEnergy.vi* in *symbol_timing.vi*, you should test your code by adding a small delay to your channel. Delay in the channel model can be modified using the *channel model parameters* control on the front panel of *Simulator.vi*. Note that delay in the channel model is bounded in *channel.vi* (i.e., you will not be able to add an arbitrarily large delay to the channel). You can use the front panel controls to modify the symbol timing recovery method used in the simulator. After choosing the STR method and modifying the delay, you should notice the estimated offset generated by the receiver changes as you vary delay. This integer offset value includes delay introduced by pulse shaping and matched filtering.

The simulator outputs the estimated delay of the channel (i.e., after correcting for any offset caused by filtering) as an error statistic. The VI *simulator.vi*

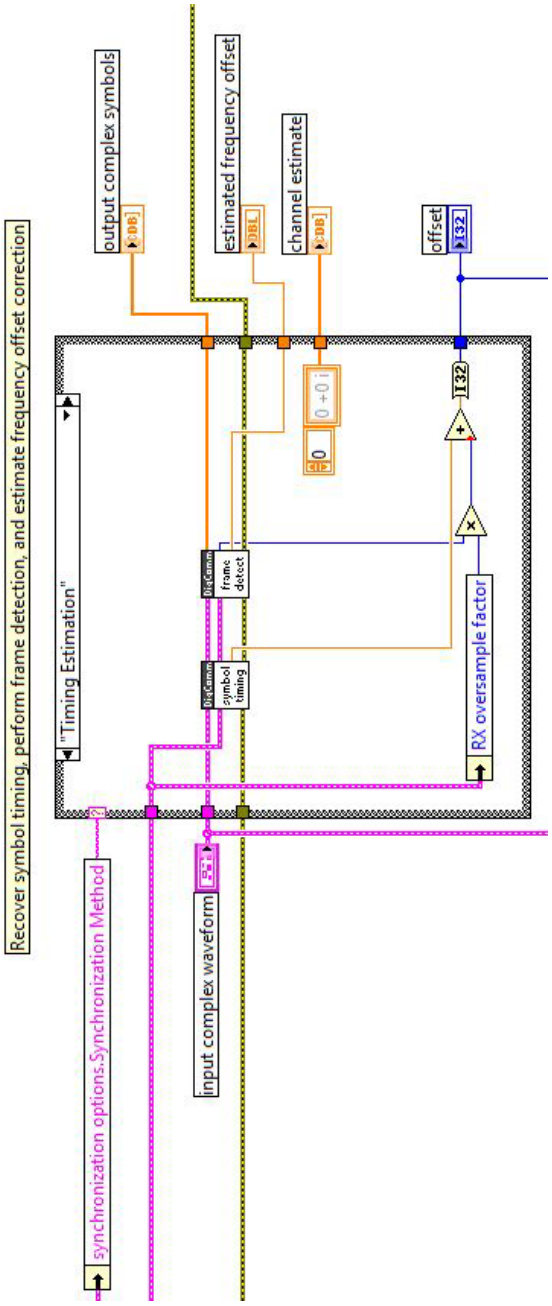

Figure 3: Block diagram of *synchronize.vi*.

compares this estimated delay with the actual delay and computes the mean square error static, which has been normalized by symbol period T_s

$$\epsilon[N] = E\left\{\left\|(\hat{\tau}(N) - \tau_d)/T_s\right\|^2\right\}, \tag{5}$$

where N is the oversample factor at the receiver, $\hat{\tau}(N)$ is the estimated delay of the channel computed at the receiver[1], and τ_d is the actual delay of the channel.

Plot a graph of the timing error statistic $\epsilon(N)$ versus oversample factor N, using the parameters below. Since your channel will have no noise, you will only need to run a single iteration of the simulator to gather the timing error statistic. Note when modifying oversample factor N, symbol period T_s must remain constant (i.e., since $T_s = NT_N$, you must also modify the sampling rate at the receiver appropriately).

- Delay $\tau_d = 0.34T_s$
- Oversample factor $N = 2, 4, 6, \ldots, 20$
- Channel Model: $h = 0.75e^{j\pi/4}$ (i.e., narrowband channel)
- Noise Power = $-\infty$ dB

In summary:

- After building *align_ELgate.vi* and *align_MaxEnergy.vi*, place these VIs into your simulator (i.e., inside *symbol_timing.vi*).
- Modify the symbol timing recovery method and the delay introduced by the channel from the front panel of *Simulator.vi*.
- After verifying the functionality of your code, observe how these algorithms perform in the presence of a narrowband channel with AWGN. In the presence of these impairments, observe how the packet length affects the performance of the algorithms you have implemented.
- Using delay $\tau_d = 0.34T_s$ and the parameters listed above, plot the timing error statistic $\epsilon(N)$, as defined in Eq. (5), for oversample factors $N = 2, 4, 6, \ldots, 20$ (i.e., even integers up to 20). Use a logarithmic scale for timing error and a linear scale for N.

[1] Estimated channel delay $\hat{\tau}(N) = \dfrac{\hat{k}T_s}{N}$, where $\hat{k}$ is the integer offset estimated at the receiver (after correcting for delay due to filtering), T_s is the symbol period, and N is the oversample factor at the receiver

Pre-Lab Turn In

1. Submit your implementation of *align_Elgate.vi* and *align_MaxEnergy.vi*.

   Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section along with your plot of timing error $\epsilon(N)$ versus oversample factor N.

3 Lab Experiment

In this lab you will run your implementations of *align_ELgate.vi* and *align_MaxEnergy.vi* over a real wireless link. Using your code from the prior labs in conjunction with the new blocks you have created, you will complete the framework for the transmitter and receiver blocks in lab. This framework will be provided to you as in previous labs.

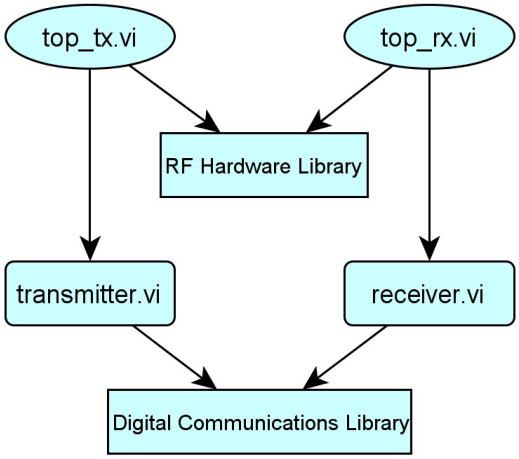

Figure 4: Hierarchy of code framework used in lab.

Insert your code into this framework as you did in the pre-lab. In the pre-lab you observed how timing error varies with the oversampling factor N at the receiver. Now you will observe this relationship over a real wireless link.

Set up the following parameters in your baseband system.

- Packet length = 500 bits
- Modulation type = QPSK

- Pulse shaping filter = Root Raised Cosine
- Filter parameter = 0.5
- Filter length = 8
- Channel model: $h = 0.75e^{j\pi/4}$ (i.e., narrowband channel)
- Noise power = $-$Inf dB
- TX sample rate = 20 MSamp/sec
- TX oversample factor = 20
- RX sample rate = 2 MSample/sec
- RX oversample factor = 2
- Symbol timing recovery method = Max Energy

Also make sure the following RF parameter is set up appropriately (using the HW parameters control accessible from the front panel of *top_rx.vi*):

- Capture time = 400 μsec

Questions

Answer the following questions about the digital communication system:

1. What is the symbol rate of the system based on the parameters above?

2. As you did in the pre-lab, vary the oversample factor N at the receiver. Observe the general shape of the received constellation for a number of packets (i.e., to eliminate any variance that might be caused by errors in frequency offset correction or channel estimation). Observe how the constellation changes when $N = 2, 4, 10$, and 20.

   Based on what you've observed and what you learned in the pre-lab, describe how the relationship between sampling error and oversample factor N manifests itself on the signal constellation of *top_rx.vi*.

3. For each of the oversample factors in the previous question (i.e., $N = 2, 4, 10, 20$), specify what value you set the *RX sample rate* to on the front panel of *top_rx.vi*.

Lab Turn In

Demonstrate to the instructor that your code is working properly. You will be required to identify that a sampling error has occurred and qualitatively support your claim based on the visualization tools available in *top_rx.vi*. You will then need to show the instructor how the performance of your system changes as you vary oversample factor N at the receiver. Also answer all of the questions above. You will be required to submit these answers in a lab report.

DIRECTIONS FOR LAB SUBMISSION

Your lab report should accomplish the following tasks.

1. Answer all questions from the lab experiment (i.e., do not reanswer pre-lab questions).
2. Discuss any problems you might have encountered and how you overcame these obstacles.
3. What underlying assumption about the noise in the system has been made, but not stated explicitly in this lab? Explain why this assumption is important for the maximum output energy maximization problem. *Hint: This is a property of AWGN.*
4. Air any grievances about the structure, content, or workload for this lab.

Lab 4: Channel Estimation & Equalization

Summary

In Lab 4 you considered a communication system using a narrowband channel model. This scalar channel model does not reflect the frequency selective nature of most realistic wireless channels. This lab will introduce digital transmission and reception in a frequency selective channel, where multiple paths in the propagation environment create distortions in the transmitted. Correcting for distortions introduced by the channel requires more complicated receiver processing. In this lab you will implement additional features in the transmitter and receiver to mitigate channel distortions. The new transmitter, channel, and receiver structure are illustrated in Figure 1.

The main concepts explored in this lab are linear least squares estimation and linear equalization. After reviewing these concepts you will build the channel estimation and equalization blocks of the model in Figure 1. You will verify the functionality and correctness of your code in a simulator. Then, you will implement your code in lab using the USRP hardware to see how your code works over a real wireless link.

The goal of this lab is to understand the need for channel equalization and to understand basic algorithms for channel estimation and channel equalization. For the pre-lab submission, you have to turn in the three VIs described in Section 2 (*LLSE.vi*, *toeplitz.vi*, and *direct_equalizer.vi*). Furthermore, you have to submit the answers to questions in Section 2.

1 Background

In wireless communication systems, the signal encounters a effects in the propagation environment including reflections, scattering, diffraction, and path-loss. The different mechanisms of propagation create multiple propagation paths between the transmitter and receiver. For example, if there are two propagation paths, the receiver may observe the signal

$$z(t) \quad = \quad \alpha_0 e^{j\phi_0} x(t-\tau_0) + \alpha_1 e^{j\phi_1} x(t-\tau_1) + v(t), \qquad (1)$$

which consists of two different delayed, attenuated, and phase shifted versions of the signal $x(t)$. The delays τ_0 and τ_1 are determined by the propagation time of the paths, which would generally be different. If the difference $\tau_1 - \tau_0$ is significant (at least a fraction of a symbol period T) then the received signal will experience intersymbol interference. This can not be corrected using the

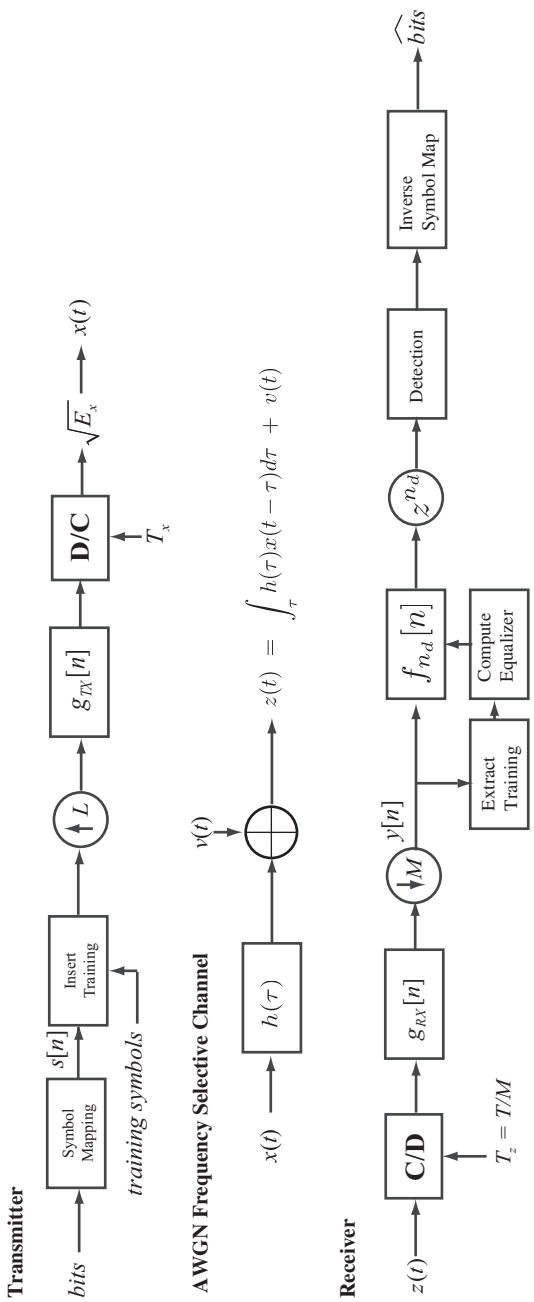

Figure 1: The system under consideration in this lab. The transmitter uses digital pulse-shaping and upsampling to create the transmit waveform. A known training sequence is inserted into the transmitted signal. The baseband signal encounters a frequency selective channel and additive white Gaussian noise at the receiver. The receiver uses digital pulse-shaping and downsampling, followed by linear equalization and detection, to find a good guess of the transmitted symbols. The linear equalizer is computed using the known training sequence in the received signal. An advance is applied after the equalization to correct for delays.

symbol synchronization techniques of Lab 3 since there is a sum of two different received signals.

In general, there may be many propagation paths between the transmitter and the receiver. A good generalization of Eq. (1) is

$$z(t) \quad = \quad \int_{\tau} h_e(\tau) x(t-\tau) d\tau + v(t), \tag{2}$$

where $h_e(\tau)$ is the baseband frequency selective channel. For complex pulse amplitude modulations, the channel distorts the pulse-shaping function. Consider the received signal after matched filtering and down-sampling. Using $\star$ for convolution define $h(t) := \sqrt{E_x} h_e(t) \star g_{\text{tx}}(t) \star g_{\text{rx}}(t) \star x(t)$ and let $h[n] = Th(nT)$ be the sampled version of the channel. Then

$$\begin{aligned} y[n] \quad &= \quad \sum_m s[m] h[n-m] + v[n] \\ &= \quad h[0] s[n] + \underbrace{\sum_{m \neq 0} s[m] h[n-m]}_{\text{intersymbol interference}} + v[n]. \end{aligned} \tag{3}$$

Unless $h(t)$ is a Nyquist pulse-shape, the second term in Eq. (3) will not be zero.

There are two main challenges associated with the received signal in Eq. (3).

1. The channel coefficients $\{h[\ell]\}$ create intersymbol interference. The solution we pursue in this lab is to augment the detection procedure with a *linear equalization* step.

2. The channel coefficients $\{h[\ell]\}$ are unknown to the receiver. The solution we pursue in this lab is to *estimate* the unknown channel coefficients and use them to determine the linear equalizer, or to estimate the coefficients of the linear equalizer directly.

The steps of estimation and equalization benefit from making some additional assumptions about the propagation channel. It is reasonable to assume that the propagation channel is causal and FIR (finite impulse response). It is causal because, naturally, the propagation channel can not predict the future. It is FIR because (i) there are no perfectly reflecting environments, and (ii) the signal energy decays as a function of distance between the transmitter and receiver. Essentially every time the signal reflects some of the energy passes through the reflector thus it looses energy. Additionally as the signal is propagating, it loses power as it spreads in the environment. Multipaths that are weak will fall below the noise threshold. With the FIR assumption, it

is common to assume that the composite channel $h(t)$ is also FIR thus Eq. (3) becomes

$$y[n] = \sum_{\ell=0}^{L} s[m]h[n-\ell] + v[n]. \quad (4)$$

The unknown channel coefficients are $\{h[\ell]\}_{\ell=0}^{L}$ where L is the order of the filter.

In this lab we will employ a linear equalizer. The goal of a linear equalizer is to find a filter that removes the effects of the channel. Let $\left\{f_{n_d}[l]\right\}_{l=0}^{L_f}$ be an FIR equalizer. The equalizer will be applied to the received signal so that

$$r[n] = \sum_{\ell=0}^{L_f} f_{n_d}[\ell]y[n-\ell] \approx \hat{s}[n-n_d], \quad (5)$$

where n_d is the equalizer delay and is generally a design parameter. Generally allowing $n_d > 0$ improves performance. The best equalizers consider several values of n_d and choose the best one.

1.1 Linear Least Squares

The main mathematical technique that will be used in this lab is known as linear least squares, which will be used both for estimating the channel and for computing the equalizer.

Let $\mathbf{A}$ denote a $N \times M$ matrix. The number of rows is given by N and the number of columns by M. If $N = M$ we say that the matrix is square. If $N > M$ we call the matrix tall and if $M > N$ we say that the matrix is fat. We use the notation $\mathbf{A}^T$ to denote the transpose of a matrix, $\mathbf{A}^*$ to denote the Hermitian or conjugate transpose, and $\mathbf{A}^c$ to denote the conjugate of the entries of $\mathbf{A}$. Let $\mathbf{b}$ be a $N \times 1$ dimension vector. The 2-norm of the vector is given by $\|\mathbf{a}\| = \sqrt{\sum_{n=1}^{N} |b_n|^2}$.

Consider a collection of vectors $\{\mathbf{x}_n\}_{n=1}^{K}$. We say that the vectors are linearly independent if there does not exist a set of nonzero weights $\{\alpha_n\}$ such that $\sum_n \alpha_n \mathbf{x}_n = \mathbf{0}$. In $\mathbb{C}^N$ and $\mathbb{R}^N$, at most N vectors can be linearly independent. We say that a square matrix $\mathbf{A}$ is invertible if the columns of $\mathbf{A}$ (or equivalently the rows) are linearly independent. If $\mathbf{A}$ is invertible, then a matrix inverse $\mathbf{A}^{-1}$ exists and satisfies the properties $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. If $\mathbf{A}$ is tall, we say that $\mathbf{A}$ is full rank if the columns of $\mathbf{A}$ are linearly independent. Similarly, if $\mathbf{A}$ is a fat matrix, we say that it is full rank if the rows are linearly independent.

Consider a system of linear equations written in matrix form

$$\mathbf{Ax} \quad = \quad \mathbf{b} \tag{6}$$

where $\mathbf{A}$ is the known matrix of coefficients sometimes called the data, $\mathbf{x}$ is a vector of unknowns, and $\mathbf{b}$ is a known vector often called the observation vector. We suppose that $\mathbf{A}$ is full rank. First consider the case where $N = M$. Then the solution to Eq. (6) is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$.

Now suppose that $N > M$. In this case $\mathbf{A}$ is tall and the system overdetermined in general. This means there are N equations but M unknowns, thus it is unlikely (except in special cases) that an exact solution exists. In this case we pursue an approximate solution known as least squares. Instead of solving Eq. (6) directly we instead propose to find the solution to the squared error

$$\min \|\mathbf{Ax} - \mathbf{b}\|^2. \tag{7}$$

Using matrix calculus, it can be shown that the solution to this problem assuming that $\mathbf{A}$ is full-rank is

$$\mathbf{x}_{LS} \quad = \quad (\mathbf{A}^*\mathbf{A})^{-1}\mathbf{A}^*\mathbf{b}. \tag{8}$$

Note that $\mathbf{A}^*\mathbf{A}$ is a square matrix and invertible because of the full-rank assumption. We refer to $\mathbf{x}_{LS}$ as the linear least square error (LLSE) solution to Eq. (6).

We use the squared error achieved by $\mathbf{x}_{LS}$ to measure the quality of the solution. With some calculus and simplifying, it can be shown that the least squares error achieved is given by

$$\begin{aligned} J(\mathbf{x}_{LS}) \quad &= \quad \|\mathbf{Ax}_{LS} - \mathbf{b}\|^2 = \mathbf{x}^*\mathbf{A}^*(\mathbf{Ax} - \mathbf{b}) - \mathbf{b}^*(\mathbf{Ax} - \mathbf{b}) \qquad (9) \\ &= \quad \mathbf{b}^*\mathbf{b} - \mathbf{b}^*\mathbf{Ax}. \qquad (10) \end{aligned}$$

Now you will use the least-squares solution for channel estimation and equalization.

1.2 Channel Estimation

There are several different criteria for designing an estimator including the maximum likelihood criterion, minimum mean squared error, and least squares. Of these, perhaps the least squares technique is the simplest thus we will discuss it here. The added advantage of the least squares estimator is that in AWGN the least squares estimator is also the maximum likelihood estimator.

Suppose that $\{t[n]\}_{n=0}^{N_t}$ is a known training sequence. This is a set of symbols that is known a priori. Suppose that the known training symbols

are inserted into the transmitted sequence such that $s[n] = t[n]$ for $n = 0, 1, \ldots, N_t$. To solve for the estimate of the channel using the LLSE solution in Eq. (8), it is necessary to form a system of linear equations. Consider

$$y[n] = \sum_{l=0}^{L} h[l]s[n-l] + v[n],$$

where $s[n] = t[n]$ for $n = 0, 1, \ldots, N_t$. Since $s[n]$ is unknown for $n \geq N_t$ (that is the unknown data), we need to write the squared error only in terms of the unknown data. With this in mind the least squares problem is to find the channel coefficients that minimize the squared error

$$\{\hat{h}[0], \hat{h}[1], \ldots, \hat{h}[L]\} = \arg \min_{a[0],a[1],\ldots,a[L]} \sum_{n=L}^{N_t-1} \left\| y[n] - \sum_{l=0}^{L} a[l]t[n-l] \right\|^2.$$

The summation starts with $n = L$ to ensure unknown data is not included. The least squares estimator is simply a generalization of the narrowband estimator. A "straightforward" way to solve this problem is to differentiate with respect to $a^*[m]$, build a set of linear equations, and solve. An alternative approach is to build a suitable set of linear equations. This is a powerful approach to solve a large class of least squares problems.

First write the observed data as a function of unknown in matrix form

$$\underbrace{\begin{bmatrix} y[L] \\ y[L+1] \\ \vdots \\ y[N_t-1] \end{bmatrix}}_{\mathbf{y}} = \underbrace{\begin{bmatrix} t[L] & \cdots & t[0] \\ t[L+1] & \ddots & \vdots \\ \vdots & & \vdots \\ t[N_t-1] & \cdots & t[N_t-1-L] \end{bmatrix}}_{\mathbf{T}} \underbrace{\begin{bmatrix} a[0] \\ a[1] \\ \vdots \\ a[L] \end{bmatrix}}_{\mathbf{a}},$$

where we refer to $\mathbf{T}$ as the training matrix. If $\mathbf{T}$ is square or tall and full rank, then $\mathbf{T}^*\mathbf{T}$ is an invertible square matrix, where * stands for conjugate transpose (Hermitian). The tall assumption (square with equality) requires that

$$N_t - L \geq L + 1,$$

or, equivalently

$$N_t \geq 2L + 1.$$

Generally choosing N_t much larger than $L+1$ (the length of the channel) gives better performance. The full rank condition can be guaranteed by ensuring that the training sequence is persistently exciting. Basically this means that it looks random enough. Random training sequences perform well while the all constant training sequence fails. Training sequences with good correlation properties generally satisfy this requirement.

1.3 Channel Equalizer Calculation

With an estimate of the channel $\{\hat{h}[l]\}_{l=0}^{L}$ in hand, the next task is to remove the effects of this channel. In this lab we consider linear equalization that performs the operation in Eq. (5). There are several approaches for designing a linear equalizer. In this lab we compute the least-squares equalizer. The objective is to find a filter such that

$$\sum_{l=0}^{L_f} f[l]\hat{h}[n-l] \approx \delta[n-n_d]. \tag{11}$$

Except in trivial channels Eq. (11) can not be satisfied exactly. The reason is that an FIR filter requires an IIR filter. The parameter n_d in Eq. (11) is the equalizer delay and is generally a design parameter. Generally allowing $n_d > 0$ improves performance. The best equalizers consider several values of n_d and choose the best one.

A straightforward approach is the least-squares equalizer. The key idea is to write a set of linear equations and solve for the filter coefficients that ensure that the Eq. (11) minimizes the squared error. Writing Eq. (11) in matrix form

$$\underbrace{\begin{bmatrix} \hat{h}[0] & 0 & \cdots & \cdots \\ \hat{h}[1] & \hat{h}[0] & 0 & \cdots \\ \vdots & \ddots & & \\ \hat{h}[L] & & & \\ 0 & \hat{h}[L] & \cdots & \\ \vdots & & & \end{bmatrix}}_{\hat{\mathbf{H}}} \underbrace{\begin{bmatrix} f[0] \\ f[1] \\ \vdots \\ \\ \vdots \\ \\ f[L_f] \end{bmatrix}}_{\mathbf{f}} = \underbrace{\begin{bmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{e}_{n_d}} \leftarrow n_d + 1 \quad .$$

The matrix $\hat{\mathbf{H}}$ is a type of Toeplitz matrix, sometimes called a filtering matrix. Assuming that $\hat{\mathbf{H}}$ is full rank, which is guaranteed if any of the channel coefficients are nonzero, the linear least squares solution is

$$\hat{\mathbf{f}}_{n_d} = \left(\hat{\mathbf{H}}^*\hat{\mathbf{H}}\right)^{-1}\hat{\mathbf{H}}^*\mathbf{e}_{n_d}.$$

The squared error is measured as $J_f[n_d] = \|\hat{\mathbf{H}}\hat{\mathbf{f}} - \mathbf{e}_{n_d}\|^2$. The squared error can be minimized further by choosing n_d such that $J_f[n_d]$ is minimized. This is known as optimizing the equalizer delay.

The equalizer order is L_f. The choice of L_f is a design decision that depends on L. The parameter L is the extent of the multipath in the channel

and is determined by the bandwidth of the signal as well as the maximum delay spread derived from propagation channel measurements. The equalizer is an FIR inverse of an FIR filter. As a consequence the results will improve if L_f is large. The complexity required per symbol, however, also grows with L_f. Thus there is a tradeoff between choosing large L_f to have better equalizer performance and smaller L_f to have more efficient receiver implementation.

1.4 Direct Least-Squares Equalizer

Using the channel estimation and equalization methods from the previous sections requires solving two least squares estimation problems. This can be computationally expensive. Designing the equalizer estimate directly from the received sequence can be more efficient as this method only requires formulating a single least-squares estimation problem. The least-squares equalizer requires solving two least-squares problems. The first is to estimate the channel coefficients using a least squares approach; the second is to estimate the least-squares equalizer. An alternative approach is a direct solution. This means that the equalizer is found directly from the observed training data. Such an approach is somewhat more robust to noise.

Consider the received signal after linear equalization with delay n_d

$$\hat{s}[n-n_d] = \sum_{l=0}^{L_f} f_{n_d}[l]y[n-l]. \tag{12}$$

Suppose that $s[n] = t[n]$ for $n = 0, 1, \ldots, N_t$ is the known training data. Then $\hat{s}[n-n_d] = t[n-n_d]$ for $n = n_d, n_d+1, \ldots, n_d+N_t$. Rewriting Eq. (12) with knowledge of the training data

$$t[n] = \sum_{l=0}^{L_f} f_{n_d}[l]y[n+n_d-l]$$

for $n = 0, 1, \ldots, N_t$.

Now build a linear equation

$$\underbrace{\begin{bmatrix} t[0] \\ t[1] \\ \vdots \\ t[N_t-1] \end{bmatrix}}_{\mathbf{t}} = \underbrace{\begin{bmatrix} y[n_d] & \cdots & y[n_d-L_f] \\ y[n_d+1] & \ddots & \vdots \\ \vdots & & \vdots \\ y[n_d+N_t-1] & \cdots & s[n_d+N_t-L_f] \end{bmatrix}}_{\mathbf{Y}_{n_d}} \underbrace{\begin{bmatrix} f_{n_d}[0] \\ f_{n_d}[1] \\ \vdots \\ f_{n_d}[L_f] \end{bmatrix}}_{\mathbf{f}_{n_d}}. \tag{13}$$

Solving under the assumption that $\mathbf{Y}$ is full rank, which is reasonable in the presence of noise, the least squares solution is

$$\hat{\mathbf{f}}_{n_d} = \left(\mathbf{Y}_{n_d}^* \mathbf{Y}_{n_d}\right)^{-1} \mathbf{Y}_{n_d}^* \mathbf{t}. \quad (14)$$

The squared error is measured as $J_f[n_d] = \|\mathbf{t} - \hat{\mathbf{Y}}_{n_d}\hat{\mathbf{f}}_{n_d}\|^2$. The squared error can again be minimized further by choosing n_d such that $J_f[n_d]$ is minimized.

Note that to ensure that $\mathbf{Y}$ is square or tall requires $L_f \leq N_t - 1$. Thus the length of the training determines the length of the equalizer. This is a major difference between the direct and indirect methods. With the indirect method, an equalizer of any order L_f can be designed. The direct method, on the other hand, avoids the error propagation where the estimated channel is used to compute the estimated equalizer. Note that with a small amount of training the indirect method may perform better since a larger L_f can be chosen while a direct method may be more efficient when N_t is larger.

2 Pre-Lab

You have learned about two equalizer estimation methods, indirect and direct. In this lab you will be implementing the direct least-squares equalizer described in the last section. To do this you will need to build a number of

Table 1: Description of *LLSE.vi*

LLSE.vi - for linear system $\mathbf{Ax} = \mathbf{b}$, as in Eq. (6), compute linear least squares estimate of parameter $\mathbf{x}$			
INPUTS	*A*	2-D array of CDB (complex doubles)	*m* x *n* matrix
	b	1-D array of CDB	length *m* vector
OUTPUTS	*x.estimate*	1-D array of CDB	linear least squares estimate of parameter $\mathbf{x}$, as defined by Eq. (8)
	least squared error	DBL (double)	squared error of estimate, $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|^2$
	error code	I32 (integer)	error code generated by *Inverse Matrix.vi*

Table 2: Description of *toeplitz.vi*

toeplitz.vi - generate an m-by-n Toeplitz structured matrix from the first row (length n) and column(length m) of the matrix			
INPUTS	*row*	1-D array of CDB (complex doubles)	first row (length n) of the matrix
	column	1-D array of CDB	first column (length m) of the matrix
OUTPUTS	*Toeplitz array*	2-D array of CDB	m-by-n Toeplitz structured array

auxiliary subVIs (i.e., *LLSE.vi* and *toeplitz.vi* described in Tables 1 and 2). You will then use these VIs in your implementation of the direct least-squares equalizer in *direct_equalizer.vi*. After building all of your VIs, you will revise the *equalizer.vi* called by *receiver.vi* and use the simulator provided to you and verify your code as you have done in the previous labs.

You can use the *error code* output of *LLSE.vi* to find out if there were any errors in computing the LLSE solution. For example, *error code* can be used to indicate an error when the matrix ($\mathbf{A}^*\mathbf{A}$) is singular.

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break.

After building these VIs, it is important that you verify their functionality before proceeding as you will need to use them in your implementation of *direct_equalizer.vi*. To verify the functionality of *LLSE.vi*, do the following:

- Let **A** be a 3x5 matrix of all zeros except for its major diagonal, which

is all ones:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

- Let **b** be any vector of length 5, such that the last two positions of **b** are zero (i.e., $\mathbf{b} = [b_1\ b_2\ b_3\ 0\ 0]^T$ contains three nonzero values in its first three positions).

- Verify that *LLSE.vi* computes *x.estimate* to be the first three elements of **b**.

Also verify that your VI returns an error when **A** is not full rank. To test this, replace the last column of **A** with zeros and repeat the above test (i.e., checking to see that an error has occurred). Next, verify your implementation of *toeplitz.vi* is working by visual inspection of *Toeplitz array*. This matrix should have constant negative-sloping diagonals.

To receive full credit you must correctly implement both *LLSE.vi* and *toeplitz.vi*. However, if you are unable to do so, you may use the equivalent VIs from *digital_comm.llb* to implement direct least-squares equalizer estimation.

Next, you will build *direct_equalizer.vi*, which will be inserted inside *equalizer.vi* as shown in Figure 2. This block diagram shows how *equalizer.vi* implements the delay optimization for both direct and indirect equalizer estimation.

To build *direct_equalizer.vi*, implement the direct least-squares estimation solution in Eq. (14). You should use your implementations of *toeplitz.vi* and *LLSE.vi* to formulate and solve the LLSE problem of Eq. (14) in *direct_equalizer.vi*. These subVIs should simplify your code significantly. Be careful when generating the matrices of Eq. (14) as small errors in the formulation can lead to critical problems.

Table 3 describes the details of *direct_equalizer.vi*. Notice the VI takes an explicit *equalizer delay* input. Do not confuse this value with the *equalizer delay* parameter in the *modulation parameters in* cluster. Use the *equalizer delay* input to form the received signal matrix of Eq. (13) (i.e., do not use the delay parameter from the *modulation parameters in* cluster).

To build the training vector **t** of Eq. (14) use *Training Sequence*, a 1-D array of CDB, from the *modulation parameters in* cluster. You will only use the first half of *Training Sequence* to generate **t**, since the training sequence from the *modulation parameters in* cluster is composed of two repeated sequences (i.e., first half of *Training Sequence* is the same as the second half). This

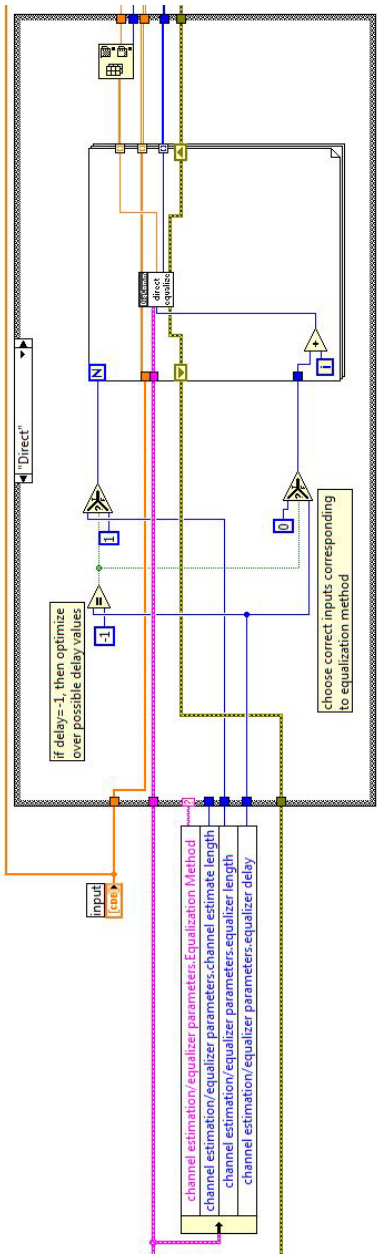

Figure 2: Block diagram of *equalizer.vi*.

Table 3: Description of *direct_equalizer.vi*

direct_equalizer.vi - solves for the least-squares solution of the equalizer directly from the received sequence as in Eq. (14)			
INPUTS	*input*	1-D array of CDB	received sequence after frame detection
	equalizer delay	I32 (integer)	filter delay parameter n_d
OUTPUTS	*filter estimate*	1-D array of CDB	solution to LLSE of Eq. (14)
	mean square error	DBL	mean square error of estimate, $\|\mathbf{Y}_{n_d}\hat{\mathbf{f}}_{n_d} - \mathbf{t}\|^2$
	actual equalizer delay	I32 (integer)	delay parameter modulo $L_f + 1$

means N_t, the length of $\mathbf{t}$, will be one-half the size of the training sequence array from the *modulation parameters in* cluster.

In summary:

- Build *LLSE.vi* using the linear algebra tools available in the **Linear Algebra** tool palette. Be aware that these polymorphic VIs operate in a number of modes and may require you to specify the use of complex inputs. The **Linear Algebra** tool palette can be accessed from **Mathematics⇒Linear Algebra**.
- Build *toeplitz.vi* using the array tools available in the **Array** palette. Be sure to verify your code before using this VI in *direct_equalizer.vi* as small errors in formulation can lead to critical problems.
- After building *direct_equalizer.vi* using the previously mentioned VIs, insert your code into *equalizer.vi*, which can be accessed from the block diagram of *receiver.vi*

Questions

Answer the following questions, and submit the answers as part of your prelab.

1. In your implementation of *toeplitz.vi* you were required to build a Toeplitz matrix given the initial row and column of the matrix. Notice

the first element of *row* and *column* should be equal. What will your VI do if the initial element of each array is different?

2. Test your channel equalizer algorithm using the channel $h[0] = 1$, $h[1] = 0.35e^{j\pi/4}$. You can modify the length of the equalizer from the front panel of the simulator. In the absence of noise, what happens to the received signal constellation when you set the equalizer length to one? Describe what happens to the constellation as you vary the equalizer length from one to six.

3. Using the same channel, observe how the bit-error rate performance of your equalizer varies with SNR for various equalizer lengths. Plot average BER as a function of SNR for $L_f + 1 = 1$ and $L_f + 1 = 6$. Vary SNR from 0 dB to 14 dB in increments of 2 dB. Use the default value for any parameter not specified below.

   - Modulation type = QPSK
   - Packet length (bits) = 500
   - Equalization method = Direct
   - Equalizer length ($L_f + 1$) = 1,6
   - ISI Channel = $\{h[0] = 1, h[1] = 0.35e^{j\pi/4}\}$

   Remember if signal power is held constant, then decreasing noise power, N_0, is equivalent to increasing SNR. To observe errors at high SNR you will need to run significantly more iterations in the simulator. Use a logarithmic scale for BER and a dB scale for SNR in your plot.

   Also, on the same graph, plot BER as a function of SNR for $L_f + 1 = 1$ in an AWGN channel for the same range of SNR values. For this curve, you will only need to plot error rates in excess of 10^{-6}.

Pre-Lab Turn In

1. Submit your implementation of *LLSE.vi*, *toeplitz.vi*, and *direct_equalizer.vi*. Remember, you will be penalized if you do not wire the error cluster inputs and outputs in your implementation of *direct_equalizer.vi*.

   Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section.

3 Lab Experiment

In this lab you will run your implementation of *direct_equalizer.vi* over a real wireless link. Using your code from the prior labs in conjunction with the new blocks you have created, you will complete the framework for the transmitter and receiver blocks in lab. This framework will be provided to you as in previous labs.

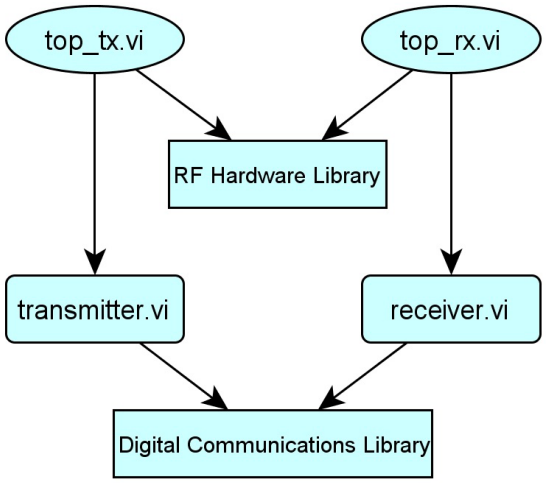

Figure 3: Hierarchy of code framework used in lab.

Insert your code into this framework as you did in the pre-lab. In the pre-lab you observed the BER performance of the direct least-squares equalizer over a multipath channel. Now you will observe how a multipath channel impacts the constellation of the received signal. You will also learn about the power-delay profile of a multipath channel.

3.1 Narrowband Channel

In this part of the lab experiment you will observe how the channel of a real wireless link impacts the received QAM constellation. To begin, setup the following parameters in your system using the appropriate controls in *top_rx.vi* and *top_tx.vi*:

- Packet length = 500 bits
- TX sample rate = 20 MSamp/sec
- TX oversample factor = 200

- RX sample rate = 2 MSamp/sec
- RX oversample factor = 20
- Equalizer length = 1
- Capture time = 3.5 msec

Use the default values for any parameters not listed above. Next, in order to observe the impact of the wireless channel on your received QAM constellation you will need to rewire part of the block diagram in *receiver.vi*. As shown in Figure 4, rewire the wire labeled **"recovered symbols"** so it is connected to the output of *synchronize.vi* and not *strip_control.vi*. This will allow you to observe the received constellation prior to equalization. Notice that since this modification bypasses *strip_control.vi* it will plot the entire received sequence (i.e., including training data and any additional symbols or zeros at the end of the array).

Questions

After setting up the appropriate parameters and modifying *receiver.vi* as previously described, run your system and observe how the narrowband channel alters your received constellation.

1. What is the symbol rate of your system?
2. What is the passband bandwidth of your system?
3. Based on your observations, describe the impairments imparted to the received constellation. You may need to "auto-scale" the axes of your constellation in *receiver.vi* in order to observe the effects of the narrowband wireless channel.

Submit your answers to these questions as part of your lab report.

3.2 Wideband Channel

In this part of the lab experiment you will learn about the power-delay profile of the channel. The power-delay profile of a continuous-time channel $h(\tau)$ is

$$P(\tau) = E\left\{|h(\tau)|^2\right\}, \tag{15}$$

where $E\{\cdot\}$ is the expectation operator. The discrete-time equivalent of the power-delay profile can be computed from the estimated channel $\hat{h}[n]$ (i.e., $P[n] = E\{|\hat{h}[n]|^2\}$). In this part of the lab experiment you will study the power-delay profile of a wideband system.

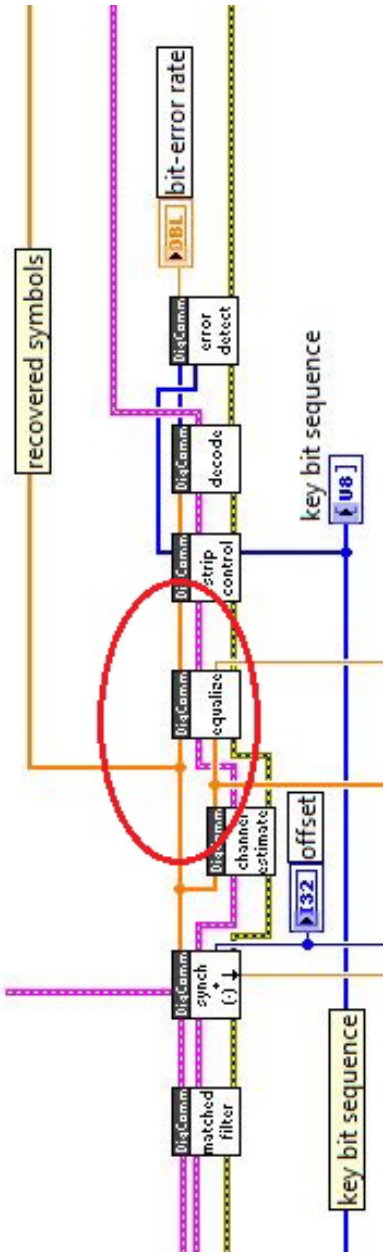

Figure 4: Modification of *receiver.vi* needed to observe constellation prior to equalization.

Set up the following parameters in your system:

- Packet length = 500 bits
- TX sample rate = 20 MSamp/sec
- TX oversample factor = 4
- RX sample rate = 10 MSamp/sec
- RX oversample factor = 2
- Channel estimate length = 6
- Equalizer length = 6
- Capture time = 80 μsec

Use the default values for any parameters not listed above. You will now start the transmitter of your system (i.e., *top_tx.vi*). Place your antennas at an elevated height so they will be free of obstructions and able to capture a large portion of the multipath energy in your wireless environment. Begin receiving packets using *top_rx.vi*. Record the estimated channel $\hat{h}[n]$ for at least 5 of the transmitted packets. Discard any measurements in which the bit-error rate of the received packet exceeds 0.10; typically BER exceeds 10^{-1} when errors in frame detection occur. You should also disable frequency offset correction from the front panel of *top_rx.vi*, as this can lead to additional unwanted errors. Also, be sure to rewire the wire labeled **"recovered symbols"** so it is connected to the output of *strip_control.vi* and not *synchronize.vi* (in contrast to the previous section).

Questions

Answer the following questions about this part of the lab experiment.

1. What is the symbol rate of your system?
2. What is the passband bandwidth of your system?
3. Based on the data you have collected in this part of the experiment, make a plot of the power-delay profile of the wideband wireless link in lab. Discard any outlying observations that might have been caused by errors in frame detection or noise. Average the power-delay profile over the remaining observations.

Submit your answers to these questions along with your plot of power-delay profile as part of your lab report.

3.3 Lab Turn In

Demonstrate that your code is working properly. Show that your code works for a variety of equalizer lengths ($L_f + 1 = 1$ and $L_f + 1 = 6$).

DIRECTIONS FOR LAB SUBMISSION

Your lab report should accomplish the following tasks.

1. Answer all questions from the lab experiment (i.e., do not reanswer pre-lab questions).
2. Discuss any problems you might have encountered and how you overcame these obstacles.
3. In this lab you observed how your code operates over a narrowband channel and a wideband channel. The following questions will explore the differences between these two channels.
   (a) What is the symbol period of the narrowband system in Section 3.1?
   (b) What is the symbol period of the wideband system in Section 3.2?
   (c) If the continuous-time channel $h(\tau)$ has nonzero support for $\tau \in [0, 2.5\ \mu\text{sec})$, then how many nonzero taps will $h[n]$, the equivalent discrete-time channel, have in the narrowband and wideband system respectively?
   In other words, if T_{nb} and T_{wb} are the symbol periods of the narrowband and wideband systems respectively, find the cardinality (i.e., number of elements) of the set $A = \{n | h[n] \neq 0\}$ for each system. *See the hint below for more on* $h[n]$.
   (d) Which of the two systems can "resolve" the channel response $h(\tau)$ more accurately?

   HINT: The equivalent discrete-time channel is $h[n] = h(nT_s)$*, where* T_s *is the symbol period of the system.*

Lab 5: Frame Detection & Frequency Offset Correction

Summary

The algorithms for channel estimation and equalization discussed in Lab 4 were formulated under the assumption that the location of the training data was known, which we call the start of the frame. In practice, due to propagation and signal processing delays, the location of the beginning of the frame is unknown. This makes it difficult to apply the channel estimation and equalization algorithms from Lab 4. Up to this point it has also been assumed that the carrier frequency of the transmitter and receiver were locked together. Even small differences, however, lead to a distortion known as carrier frequency offset. In this lab you will implement a particular method for frequency offset correction, called the Moose algorithm, and a method for frame detection based on correlation.

The frequency offset correction and frame detection algorithms you will implement in this lab will be embedded within the synchronization block of the receiver. After building the VIs needed to implement these algorithms, you will verify the functionality and correctness of your code in a simulator. Then, you will implement your code in lab using RF Hardware to see how your code works over a real wireless link.

Note that after this lab you will have considered symbol timing synchronization, frame detection, and channel estimation/equalization all as independent problems. In actuality, any number of these problems could be solved jointly (e.g., jointly solve for the symbol timing offset and channel estimate which minimizes mean-squared error). The task of joint optimization will not be explored in this lab, but you should be aware that the receiver you have been implementing in lab is not necessarily the optimal one. In particular, it is common that frame synchronization is performed jointly with carrier frequency offset estimation.

For the pre-lab submission, you have to turn in the two VIs described in Section 2 (*Moose.vi* and *sliding_correlator.vi*). Further, you have to submit the answers to the questions in Section 2.

1 Background

In this section we provide some background on training sequences then review algorithms for frame synchronization and frequency offset synchronization.

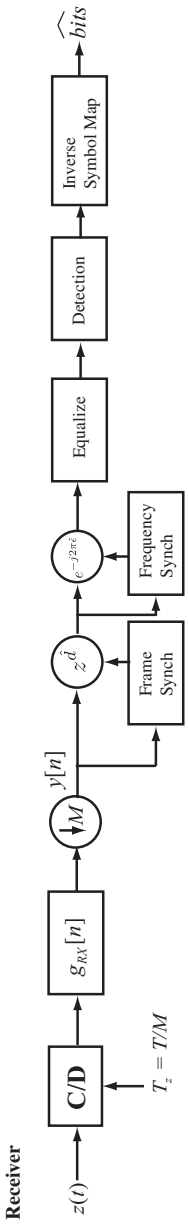

Figure 1: The receiver considered in this lab. The new synchronization functionality is introduced after downsampling and before the equalization. The first step is determining the location of the training sequence, known as frame synchronization. The second step is estimation and correction of the carrier frequency offset, known as carrier frequency synchronization. The linear equalizer block includes the channel estimation and equalization functions from Lab 4.

1.1 Training Sequences

Up to this point we have assumed that a training sequence has been provided at both the transmitter and receiver. Not all training sequences are created equal. Without a good training sequence, the estimation techniques in the labs would perform poorly. This section discusses some desirable properties of training sequences and introduces one family of sequences known as Barker codes. Many other sequences are used in research literature and in commercial wireless systems as well (e.g., Frank sequences, Zadoff-Chu sequences, or Gold sequences).

Training sequences serve as a known reference signals, which can be used for many purposes at the receiver. In these labs, we use training sequences for two main purposes: (1) synchronization and (2) channel estimation. In general, *good* training sequences for synchronization possess strong autocorrelation properties. In Lab 4, the training sequence needed to be suitably designed in order to provide sufficient rank properties. Training sequences with strong autocorrelation properties can satisfy these rank properties. In this lab, you will use such a training sequence to create a correlation based detector that finds the beginning of a frame.

Barker codes are one set of sequences with good aperiodic autocorrelation and rank properties. A length N_t Barker sequence $\{a_k\}_{k=1}^{N_t}$ is a sequence of values, ± 1, such that

$$\left| \sum_{i=1}^{N_t-k} a_i a_{i+k} \right| \leq 1, \tag{1}$$

for all $1 \leq k \leq N_t$. Because of this strong autocorrelation property, Barker sequences are commonly used in digital communication systems that employ spread spectrum techniques. They have been used for example in IEEE 802.11 and IEEE 802.11b. Table 1 has a list of known Barker codes[1]. The complete set of Barker sequences is the closure of this set of codes under reversal and negation.

The known binary sequences that satisfy Eq. (1) are listed in Table 1. It may happen that longer training sequences are required in a given application. Remember that longer training sequences give better estimation performance. One approach is to relax the condition in Eq. (1) to allow for large cross-correlation values or the binary restriction and thus look for another sequence family. Another approach is to concatenate several Barker sequences together. This will give higher cross-correlation peaks, but provides some other properties that will be exploited for frequency offset estimation.

[1] Shorthand notation is used in this table. "−" and "+" represent −1 and +1 respectively.

Code Length	Barker Sequence
2	$[-+, --]$
3	$[--+]$
4	$[-+--, -+++]$
5	$[---+-]$
7	$[---++-+]$
11	$[---+++-++-+]$
13	$[-----+--+-+-+-]$

Table 1: Barker Sequences

In the lab we will use a training sequence that consists of four length 11 Barker sequences prepended to the data segment of the packet. This means that you may treat the training sequence as a concatenation of two length 22 training "sub"-sequences (i.e., the complete training sequence is 44 symbols, and the first and second half are the same).

1.2 Frame Synchronization

The purpose of frame synchronization is to resolve multiple symbol period delays, determining what we call the beginning of the frame. Essentially the objective is to find a known reference in the transmitted signal so that subsequence receiver operations can proceed like channel estimation and equalization. There is a hugh theory surrounding frame synchronization. We consider one approach based on correlation properties of the training signal. To develop the frame synchronization algorithm we neglect the presence of carrier frequency offset in this section.

Consider a flat-fading channel where symbol synchronization has already been employed to give

$$y[n] = hs[n-d] + v[n]$$

after the matched filter and down-sampling at the receiver where h is an unknown complex channel coefficient and d is an unknown frame offset. Suppose that data is transmitted in frames consisting of a length N_t training signal followed by $P - N_t$ data symbols. Suppose that $\{t[n]\}_{n=0}^{N_t}$ is the training sequence known at the receiver. Since the training sequence has good correlation properties, we correlate the received signal with the training signal to compute

$$R[n] = \left| \sum_{k=0}^{N_t-1} t^*[k]y[n+k] \right|^2 \tag{2}$$

and solve for the frame offset as

$$\hat{d} = \max_n R[n]. \tag{3}$$

The correlation in Eq. (2) results in two terms

$$R[n] \quad = \quad \left| \sum_{k=0}^{N_t-1} t^*[k] h s[n-d] + \sum_{k=0}^{N_t-1} t^*[k] v[n] \right|^2 .$$

Because the noise is zero mean, the second term should be approximately zero if the training sequence is large enough. This leads to

$$R[n] \quad \approx \quad \left| \sum_{k=0}^{N_t-1} t^*[k] h s[n+k-d] \right|^2 = |h|^2 \left| \sum_{k=0}^{N_t-1} t^*[k] s[n-d] \right|^2 . \tag{4}$$

Assuming that the data is different than the training sequence, the correlation peak should occur at the location of the training data, assuming that the training data is contained in the window of observation samples.

Now suppose that the channel is frequency selective and

$$y[n] = \sum_{\ell=0}^{L} h[\ell] s[n-\ell-d] + v[n] = h[\ell] \star s[n-d] + v[n],$$

where we use $\star$ to denote convolution. In this case, the received correlation $R[n]$ neglecting noise is

$$R[n] \quad \approx \quad \left| h[\ell] \star t^*[-n] \star s[n-d] \right|^2 . \tag{5}$$

The effect of the channel is to "smear" the correlation peak. Correlation based frame detection still works, but it becomes less accurate as the peak may shift due to the channel. In practice the peak value of $R[n]$ may be shifted by an offset $\hat{d} = \max_n R[n] - \Delta$ for some $\Delta > 0$ typically $\Delta < L/4$. Equalization takes care of the residual offset.

1.3 Frequency Offset Estimation

Passband communication signals are used in wireless communication systems. A passband signal has energy that is nonzero for a frequency band concentrated about some carrier frequency f_c. At the transmitter the baseband signal is upconverted to a passband signal while at the receiver a passband signal is downcoverted to a baseband signal. In most systems the carrier frequency used for upconversion at the transmitter does not match the car-

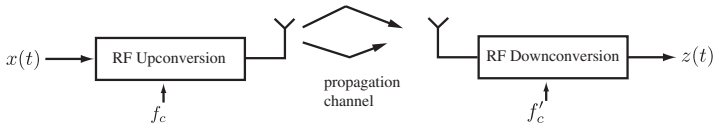

Figure 2: Frequency offset between transmitter and receiver.

rier frequency at the receiver, as illustrated in Figure 2. This creates what is known as a carrier frequency offset.

From a baseband signal perspective, the problem of carrier frequency offset can be visualized using the signal model in Figure 3 where the carrier offset is $f_o = f_c - f_c'$. The effect of the offset is to rotate the received signal by $\exp(j2\pi f_o t)$. The larger the offset, the faster the received signal rotates. The carrier offset complicates most of the receiver processing functions including frame synchronization, channel estimation, and equalization.

If the carrier offset is small and sufficient sampling is employed, then the discrete-time signal at the receiver after matched filtering and downsampling can be written as

$$y[n] \quad = \quad e^{j2\pi\epsilon n} \sum_{l=0}^{L} h[l]s[n-l] + v[n],$$

where $\epsilon = f_o T$. The objective of carrier frequency offset synchronization is to estimate the parameter ϵ. Given an estimate $\hat{\epsilon}$, removal of the offset amounts to performing

$$\tilde{y}[n] = e^{-j2\pi\epsilon n} y[n]. \tag{6}$$

To estimate the carrier frequency offset, we use a data aided approach proposed by Moose [7]. This method relies on a periodic training sequence and is applicable to many commercial wireless systems (e.g., IEEE 802.11a/g systems). It a self-referenced synchronization algorithm as it employs the periodic structure in the training but not the correlation properties. Consider

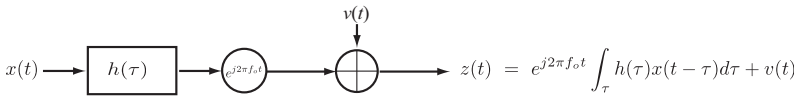

Figure 3: Frequency offset between transmitter and receiver.

a complex pulse-amplitude modulation where there is a repetition of two training sequences followed by the data. Let the training sequence start at $n = 0$. Thus

$$s[n] = s[n + N_t] = t[n]$$

for $n = 0, 1, \ldots, N_t - 1$. Note that symbols $s[n]$ for $n < 0$ and $n \geq N_t$ are unknown (they are either zero or correspond to the unknown portions of the data). For $L \leq n \leq N_t - 1$

$$\begin{aligned} y[n] &= e^{j2\pi\epsilon n} \sum_{l=0}^{L} h[l]s[n-l] + v[n] \\ y[n+N_t] &= e^{j2\pi\epsilon(n+N_t)} \sum_{l=0}^{L} h[l]s[n+N_t-l] + v[n+N_t]. \end{aligned}$$

Using the fact that $s[n + N_t] = s[n] = t[n]$ for $n = 0, 1, \ldots, N_t - 1$

$$\begin{aligned} y[n+N_t] &= e^{j2\pi\epsilon N_t} e^{j2\pi\epsilon n} \sum_{l=0}^{L} h[l]t[n-l] + v[n+N_t] \\ &\approx e^{j2\pi\epsilon N_t} y[n]. \end{aligned} \tag{7}$$

To see the significance of this result, remember that the channel coefficients $\{h[l]\}_{l=0}^{L}$ are *unknown*!

One way to solve the frequency offset estimation problem is to formulate and solve a least-squares problem. Because ϵ appears in the exponent, we solve a modified least squares problem. Consider the squared error

$$J(a) = \sum_{l=L}^{N_t-1} \|y[l+N_t] - ay[l]\|^2.$$

Using the concept of liner least squares from Lab 4, the coefficient a is

$$\hat{a} = \frac{\sum_{l=L}^{N_t-1} y[l+N_t]y^*[l]}{\sum_{l=L}^{N_t-1} |y[l]|^2}.$$

Since only the phase of $\hat{a}$ is of interest, there is no need to compute the denominator. A simple estimate of the frequency offset is then

$$\hat{\epsilon} = \frac{\text{phase} \sum_{l=L}^{N_t-1} y[l+N_t]y^*[l]}{2\pi N_t} \tag{8}$$

or

$$\hat{f}_e = \frac{\text{phase} \sum_{l=L}^{N_t-1} y[l+N_t]y^*[l]}{2\pi T N_t}, \tag{9}$$

where phase denotes the principle phase of the argument.

Thanks to the periodicity of the discrete-time exponential, the estimate of ϵ will only be accurate for $|\epsilon N_t| \leq \frac{1}{2}$ or equivalently

$$|\epsilon| \leq \frac{1}{2N_t}$$

or

$$|f_e| \leq \frac{1}{2TN_t}.$$

Choosing larger N_t improves the estimate since it results in more noise averaging but reduces the range of offsets that can be corrected. A way of solving this problem is to use multiple repetitions of a short training sequence, but this will not be exploited in this lab. An enhanced carrier frequency offset correction range can be obtained by using properties of the training sequence.

1.4 Combining Frame Detection and Frequency Offset Estimation

The frame synchronization algorithm in this lab was developed assuming there was no carrier frequency offset. The presence of frequency offset reduces the accuracy of the correlation based frame detection method. For small offsets this effect can be negligible while for larger offsets, the autocorrelation properties of the training sequence can "fall apart."

The carrier frequency offset estimation algorithm considered in this lab assumed that frame synchronization was already performed. The periodic correlation properties though could be used to perform frame synchronization as well. The observation is that the correlation peak should occur when the pair of training sequences is encountered at the receiver. A robust approach involves evaluating the received correlation in a window and solving

$$\hat{d} = \arg\max \frac{\sum_{n=L}^{N_t-1} y[n+d+N_t]y^*[n+d]}{\sqrt{\sum_{n=L}^{N_t-1} |y[n+d]|^2}\sqrt{\sum_{n=L}^{N_t-1} |y[n+d+N_t]|^2}}. \quad (10)$$

Frame synchronization with the Moose method is robust to intersymbol interference.

In the lab we will employ the training based correlation for frame synchronization followed by the Moose method for carrier frequency offset estimation. It should be clear though that other approaches are possible, including algorithms that jointly perform frame synchronization and carrier frequency offset estimation.

Table 2: Description of *Moose.vi*

Moose.vi - implements the Moose algorithm described by Eq. (9) and performs correction per Eq. (6).			
INPUTS	*input*	1-D array of CDB (complex doubles)	received sequence after frame detection
	symbol rate	DBL (double)	symbol rate of the system
	channel estimate length	I32 (integer)	$L_h + 1$
OUTPUTS	*output*	1-D array of CDB	received sequence after correcting for estimated frequency offset, i.e., $\tilde{y}[n]$ as described by Eq. (6)
	frequency offset	DBL	frequency offset estimate $\hat{f}_{\text{offset}} = \hat{\epsilon}/T$

2 Pre-Lab

In this lab you will implement the correlation based frame detector and the Moose algorithm. You will first build *Moose.vi*, that will implement the Moose algorithm. You will then build *sliding_correlator.vi*, which will implement the frame detection algorithm. This VI will also use *Moose.vi* to estimate/correct for the frequency offset. These VIs are described in Tables 2 and 3. After building your VIs, you will insert your code into *frame_detect.vi* in the simulator provided to you and verify your code using the simulator. Please name your VIs in the same manner as your previous submissions (i.e., follow the *yourname_VIname.vi* format).

After building these VIs, insert your code into the simulator provided to you. As shown in Figure 4, you will insert *sliding_correlator.vi* into the appropriate case structure in the block diagram of *frame_detect.vi*.

You can find *frame_detect.vi* in the block diagram of *synchronize.vi*. The hierarchy of these files is shown in Figure 5. After inserting your code appropriately, if you find that the simulator does not compile check to make sure that you have used the same connector topology described by *moose.vi* in the digital communication library.

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What

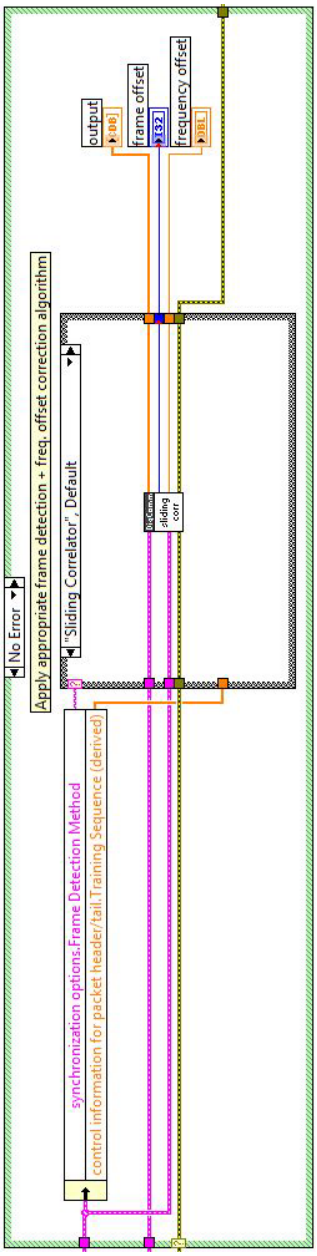

Figure 4: Block diagram of *frame_detect.vi*.

Table 3: Description of *sliding_correlator.vi*

sliding_correlator.vi - implements the correlation based detector of Eqs. (2)–(3) ; also estimates/corrects for frequency offset using *Moose.vi*.			
INPUTS	*input complex waveform*	IQ waveform cluster	received sequence after matched filter and symbol timing recovery
	correct frequency offset?	Boolean	determines whether or not your VI should correct for the frequency offset estimated by *Moose.vi*
OUTPUTS	*output*	1-D array of CDB	received sequence after correcting for estimated delay $\hat{d}$ and frequency offset (when appropriate)
	frame offset	I32	estimated delay $\hat{d}$ of Eq. (3)
	frequency offset	DBL	frequency offset estimate computed by *Moose.vi*

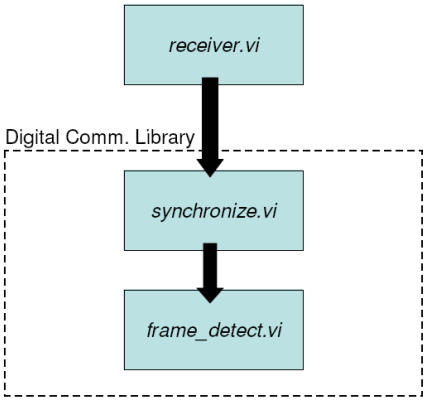

Figure 5: Hierarchy of files associated with *frame_detect.vi*.

is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation*

parameters in contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break.

After building these VIs and inserting them into the simulator as you have done in previous labs, it is important that you verify their functionality. First, verify that your correlation based detector is working by doing the following:

1. Set the *Correct Frequency Offset* control to FALSE on the front panel of the simulator. Setting this constant to FALSE tells *frame_detect.vi* not to correct for the frequency offset estimated by *Moose.vi*.

2. Set the following parameters in your simulator.

   - TX oversample factor = RX oversample factor = 10
   - TX sample rate = RX sample rate = 10 MHz
   - Packet length (bits) = 500
   - Synchronization Method = Timing Estimation
   - ISI Channel = $\{h[0] = 0.35e^{j\pi/4}\}$

   Use the default value for any parameters not specified above.

3. Set the delay of the channel to $10T_s$, where T_s is the symbol period of the system.

After running the top level of your simulator, verify that your receiver has estimated the correct delay in the channel. You can find the measured delay in the *Measured channel impairments* cluster on the front panel of the simulator. Additionally, you should verify that you have no bit errors. After you have verified that your correlation based frame detection algorithm has been implemented properly, you can now check that your implementation of the Moose algorithm is working properly by doing the following:

1. Set the *Correct Frequency Offset* control to TRUE on the front panel of the simulator Setting this constant to TRUE tells *frame_detect.vi* to correct for the frequency offset estimated by *Moose.vi*.

2. Use the same parameters to test your implementation of *Moose.vi*.

3. Set the delay of the channel to $10T_s$, where T_s is the symbol period of the system.

4. Set the frequency offset of the channel to 201 Hz.

Verify that the estimated frequency offset displayed in the *Measured channel impairments* cluster reflects the correct frequency offset. Also, you should have no bit errors. Finally, set the *Correct Frequency Offset* control back to FALSE. Verify that your code estimates the frequency offset, but does not correct for it when this Boolean control is set to FALSE.

In summary:

- After building *Moose.vi* and *sliding_correlator.vi*, insert your code into the simulator (i.e., inside *frame_detect.vi*).

  Note: You will use the unbundled parameter "Training Sequence" from the modulation parameters in cluster for the training sequence in this lab.

- Verify that your implementation of the correlation based frame detector and the Moose algorithm are working properly.

Questions

Answer the following questions, and submit the answers as part of your pre-lab.

1. Set up your simulator according to the steps used for verifying your correlation based frame detector. In particular, make sure that the control *Correct Frequency Offset* is set to FALSE. Set the noise power of your channel to 5 dB. Also, use AWGN channel for this question instead of ISI. Describe what happens to the error rate of your system when the estimate for the delay in the channel is off by more than one symbol time (i.e., when $\hat{d} \neq d$).

   Note: You may need to modify your VIs to observe this effect, or you may guess the impact from the structure of the frame detection equations.

2. Set up your simulator according to the steps used for verifying your implementation of the Moose algorithm. Set the *Correct Frequency Offset* control to FALSE. Describe what happens to the received constellation if you do not correct for the 201 Hz frequency offset.

3. Based on the system parameters described in the pre-lab, what are the range of frequency offsets that you can estimate/correct using the Moose algorithm? Since the Moose algorithm cannot estimate/correct for arbitrary frequency offsets it is often considered to perform what is known as fine frequency synchronization.

Pre-Lab Turn In

1. Submit your implementation of *Moose.vi* and *sliding_correlator.vi*. Remember, you will be penalized if you do not wire the error cluster inputs and outputs in *sliding_correlator.vi* and *Moose.vi*.

   Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section.

3 Lab Experiment

In this lab you will run your implementation of *sliding_correlator.vi* over a real wireless link. Using your code from the prior labs in conjunction with the new blocks you have created, you will complete the framework for the transmitter and receiver blocks in lab as in previous labs. Insert your code into this framework as you did in the pre-lab. Next, set up the following parameters in your system.

- TX sample rate = 4 MSamp/sec
- TX oversample factor = 20
- RX sample rate = 4 MSamp/sec
- RX oversample factor = 20
- Channel estimate length = 2
- Equalizer length = 4
- Capture time = 2 msec

Leave any unspecified parameters set to their default values. Also make sure that the *Correct Frequency Offset* control on *top_rx* is set to TRUE. You will now perform two experiments to explore frequency offsets and frame detection over a real wireless link. Before you begin, estimate the order of magnitude of the frequency offset between the transmitter and receiver of your system. Take note of this value as you may be asked about it later.

3.1 Correlation Based Frame Detection Under Large Frequency Offsets

The correlation based detector discussed in this lab can be impacted by a number of impairments, including frequency offset, as discussed previously.

Answer the following questions to explore the relationship between frequency offset and your correlation based detector.

1. Before changing any of the settings, calculate an average value (using five runs or so) of the inherent offset between the transmitter and receiver.

2. Based on the system parameters, what is the range of frequency offsets that can be estimated by your frequency offset estimation algorithm?

3. Let f_M be the maximum correctable frequency offset of your system (i.e., as calculated in the previous question). Modify the carrier frequency of your transmitter so that you will cause an effective offset of $0.80\,f_M$ at the receiver. What is the new value of the carrier frequency at your transmitter?

4. How does this frequency offset impact the performance of your correlation based frame detector (i.e., does your correlation based detector still find the start of the frame)? Describe what happens to the error rate and constellation of your system.

You will include your answers to these questions in your lab report. *Note that the frequency offset $0.80\,f_M$ is two orders of magnitude greater than the inherent offset between the transmitter and receiver.*

The correlation based detector discussed in this lab can be impacted by a number of impairments, including frequency offset, as discussed previously. Answer the following questions to explore the relationship between frequency offset and your correlation based detector.

1. Based on the system parameters, what is the range of frequency offsets that can be estimated by your frequency offset estimation algorithm?

2. Let f_M be the maximum correctable frequency offset of your system (i.e., as calculated in the previous question). Modify the carrier frequency of your transmitter so that you will cause an effective offset of $0.80\,f_M$ at the receiver. What is the new value of the carrier frequency at your transmitter?

3. How does this frequency offset impact the performance of your correlation based frame detector (i.e., does your correlation based detector still find the start of the frame)? Describe what happens to the error rate and constellation of your system.

You will include your answers to these questions in your lab report.

3.2 Self referenced frame synchronization

In Section 3.1, you saw how the correlation based detector can "break" under large frequency offsets. You will now explore another frame detection algorithm which is more robust to frequency offsets. As discussed in class, a self referenced frame synchronization method can be derived from the least-squares expression of the Moose algorithm. This frame detection performs the calculation in Eq. (10). This frame detection method has already been implemented in *digital_comm3.2b.llb*. Change the frame detection method of your system by modifying the appropriate parameter on the front panel of *top_rx.vi*. Observe how the *Self Reference Frame Synchronization* method performs under the large frequency offset computed in the last section.

Consider a simple flat fading channel model with frequency offset $f_{\text{offset}} = \epsilon/T$ (T is the symbol period of the system) such that the received sequence is given by

$$y[n] = \alpha\, e^{j2\pi\epsilon n}\, t[n - D], \tag{11}$$

where $t[n]$ is the transmitted training sequence, α is a complex scalar, and D is the delay introduced by the channel. Include the answer to the following question in your lab report:

Question: Using the simple channel model of Eq. (11), explain why the self referenced frame detector of Eq. (10) works better than the correlation based detector under large frequency offsets. You may also need to consider the mathematical description of the correlation based detector given in Eq. (2). *Hint: Notice that the training sequence is periodic (i.e., $t[n + W] = t[n]$ for all $n \in \{0, 1, \ldots, W - 1\}$). Also you should notice that the output of the frame detection algorithm should be D for this particular channel model.*

3.3 Lab Turn In

Demonstrate to the instructor that your code is working properly. Show the instructor that your correlation based detector "breaks" under a large frequency offset (as discussed in Section 3.1). Then show the instructor that under this same frequency offset, the self referenced frame synchronizer works. Also answer all of the questions above. You will be required to submit these answers in a lab report.

DIRECTIONS FOR LAB SUBMISSION

Your lab report should include the following.

1. Answer all questions from the lab experiment (i.e., do not reanswer pre-lab questions).

2. Discuss any problems you might have encountered and how you overcame these obstacles.

3. In this lab you observed how your code operates over a real wireless link. Answer the following questions about your experience in lab.

   (a) Name three impairments in a real wireless system which impact the accuracy of the correlation based detection method.

   (b) Under large frequency offsets, the correlation based detector "breaks". What property of the training sequence breaks down under large frequency offsets? *Hint: You may need to consider your answer to the question in Section 3.2*.

   (c) In the self referenced frame synchronization method, is it still critical that the periodic training sequence have good autocorrelation properties? If so, why? Use the simple channel model of Eq. (11) to help answer this question, if necessary.

Lab 6: OFDM Modulation & Frequency Domain Equalization

Summary

In this lab you will implement the key features of the orthogonal frequency division multiplexing (OFDM) multicarrier modulation technique. OFDM is a transmission technique with a special structure that permits low complexity linear equalization at the receiver. Several commercial wireless systems have adopted OFDM modulation including wireless LAN standards like IEEE 802.11g, IEEE 802.11a, and IEEE 802.11n; broadband wireless access including IEEE 802.16 (WiFi); mobile broadband wireless known as IEEE 802.20; digital video broadcasting DVB (used in Europe); as well as several releases of the 3GPP cellular standards. The OFDM system considered in this lab is illustrated in Figure 1.

Although the design of an OFDM system has several important differences from the single carrier system considered in previous labs, it must still perform many of the same functions (e.g., channel estimation, equalization, and frequency offset estimation). In this lab you will still perform all functions related to synchronization and channel estimation in the time domain using the same training sequence discussed in Lab 5. The framing structure considered in this lab is illustrated in Figure 2.

For the pre-lab submission, you have to turn in the three VIs described in Section 2 (*OFDM_modulate.vi*, *OFDM_demodulate.vi*, and *FEQ.vi*). Further, you have to submit the answers to questions in Section 2.

1 Background

1.1 OFDM Modulation

Figure 1 shows a block diagram of the basic OFDM modulator you will be implementing in this lab. OFDM is a type of digital modulation where information is modulated into discrete-time sinusoids. With OFDM, the symbols after the constellation mapping are considered to start in the frequency domain. OFDM operates on groups of symbols, the resulting group of symbols being called an OFDM symbol. To begin the explanation, we assume $K = 0$ and that there are N frequency-domain symbols in one OFDM symbol. The number of subcarriers is given by N, typically a power of 2. The transmitter operates as follows. Given $\{s[n]\}_{n=0}^{N-1}$ and cyclic prefix of length L_c, the

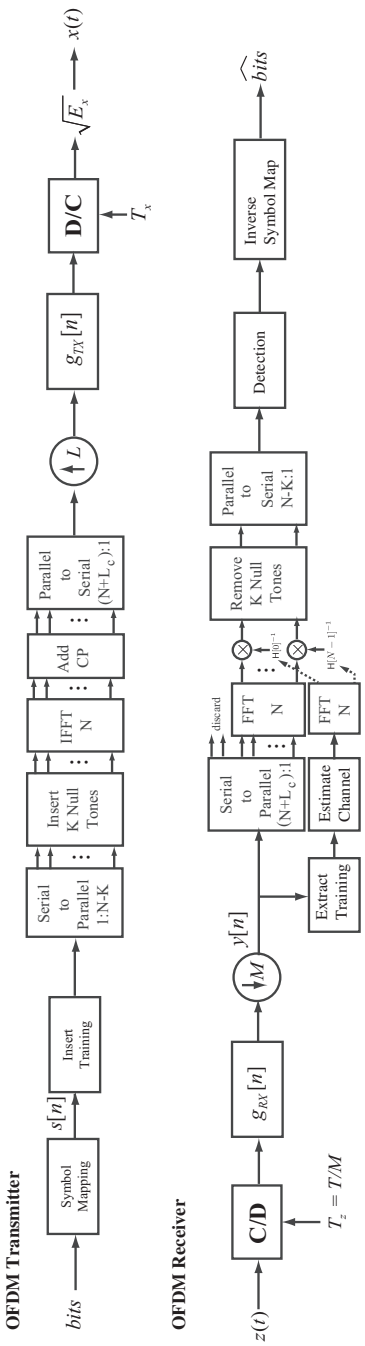

Figure 1: The OFDM system considered in this lab. The transmitter sends groups of $N-K$ symbols where N is the number of subcarriers and K the number of null subcarriers. Null symbols are inserted into groups of $N-K$ transmit symbols, which are transformed from frequency to time domain, then prepended with a cyclic prefix of length L_C at the transmitter. The receiver processes blocks of length $N+L_C$, discarding the first L_C samples of each block. Single tap equalization is performed in the frequency domain. The operations of carrier frequency synchronization and frame synchronization are omitted for simplicity.

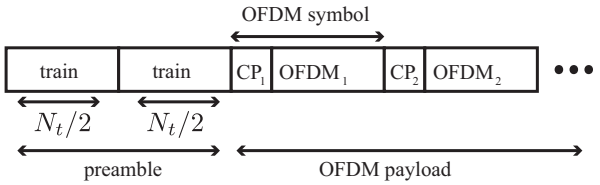

Figure 2: The framing structure considered in this lab. The preamble consists of two repeated training sequences from Lab 5 (a total of four Barker codes). The preamble is sent using the usual complex pulse amplitude modulation. Subsequent data is sent using OFDM modulation, which is a block-based modulation technique.

transmitter produces the sequence

$$w[n] = \frac{1}{N}\sum_{n=0}^{N-1} s[m]e^{j2\pi\frac{m(n-L_c)}{N}} n = 0, ..., N + L_c - 1$$

that is passed to the transmit pulseshaping filter. The samples from $n = L_c, L_c + 1, \ldots, N + L_c - 1$ are the output of the inverse discrete Fourier transform (DFT) of the input symbols $\{s[m]\}_{n=0}^{N-1}$, which could be implemented using the fast Fourier transform (FFT).

Now observe that

$$w[n] = w[n + N]$$

for $n = 0, 1, \ldots, L_c - 1$. This means that the first L_c samples of $w[n]$ are the same as the last L_c samples. The first L_c samples are known as a cyclic prefix. The length of the cyclic prefix L_c should be at least as long as L_h. This will ensure that there is no intersymbol interference between adjacent OFDM symbols (i.e., the CP "guards" against ISI). The cyclic prefix serves another important purpose. It helps convert (part of) the linear convolution into a circular convolution.

In OFDM rectangular pulse-shaping is common; windowing of the symbols is used to shape the frequency spectrum. This is not considered in this lab.

Assuming carrier frequency synchronization and frame synchronization have been accomplished, the received observes the following signal after matched filtering, symbol timing, frame synchronization, and downsampling

$$y[n] = \sum_{\ell=0}^{L} h[\ell]w[n - \ell] + v[n].$$

The receiver operates on blocks of data.

Note: This exposition implies that to recover multiple blocks of symbols (i.e., the input to the OFDM modulator), you will need to process multiple OFDM symbols, each of length $N + L_c$. Assuming that the indexing starts at the first OFDM symbol, the receiver discards the first L_c samples to form for $n = 0, 1, \ldots, N-1$ (neglecting noise) $\bar{y}[n] = y[n + L_c]$. With some mathematical manipulations it can be shown that

$$\begin{aligned}
\bar{y}[n] &= \sum_{l=0}^{L} h[l]w[n + L_c - l]n = 0, ..., N-1 \\
&= \frac{1}{N}\sum_{l=0}^{L} h[l] \sum_{m=0}^{N-1} s[m]e^{j2\pi \frac{m(n+L_c-L_c-l)}{N}} \\
&= \frac{1}{N}\sum_{l=0}^{L} h[l] \sum_{m=0}^{N-1} s[m]e^{j2\pi \frac{mn}{N}} e^{-j2\pi \frac{ml}{N}} \\
&= \frac{1}{N}\sum_{m=0}^{N-1} \left(\sum_{l=0}^{L} h[l]e^{-j2\pi \frac{ml}{N}} \right) s[m]e^{j2\pi \frac{mn}{N}}.
\end{aligned}$$

The receiver then takes the DFT (or FFT) of these samples to produce (after some simplification)

$$\begin{aligned}
\bar{Y}[k] &= DFT\,[\bar{y}[n]] \\
&= H[k]s[k] + V[k], \qquad (1)
\end{aligned}$$

where

$$H[k] = \sum_{l=0}^{L} h[l]e^{-j2\pi \frac{kl}{N}},$$

which is just the DFT of the zero padded channel. The frequency domain interpretation of OFDM comes from Eq. (1). Essentially information is sent on discrete-time sinusoids, or carriers. The information on the k^{th} discrete-time sinusoid experiences the channel response determined by $H[k]$. Equalization simply requires dividing $\bar{Y}[k]$ by $H[k]$. We refer to this as the frequency domain equalizer (FEQ).

The following parameters are useful when discussing OFDM.

- T is sample period.
- $T(N + L_c)$ is the OFDM symbol period.
- The guard interval, or cyclic prefix duration, is $L_c T$.

- The passband bandwidth is $1/T$ assuming use of a sinc pulseshaping filter.
- The subcarrier spacing $\Delta_c = \frac{BW}{N} = \frac{1}{NT}$.

The length of the cyclic prefix L_c should be the same as or exceed the order of the channel response L_h. Recall that the length of a channel of order L_h is $L_h + 1$. This will ensure that there is no intersymbol interference between adjacent OFDM symbols (i.e., the CP "guards" against ISI). The guard interval is a form of overhead. This reduces the effective data rate that is transmitted (i.e., for an OFDM system with OFDM sample rate R_s and M bits/symbol, the effective data rate $R_d < R_s$).

The guard interval serves to separate different OFDM symbols thus the name. In practice the guard interval is determined by the maximum delay spread. As the bandwidth increases, L_c must increase to compensate. The subcarrier spacing refers to the spacing between adjacent subcarriers as measured on a spectrum analyzer. The larger the N, the smaller the subcarrier spacing. The subcarrier spacing determines the sensitivity to Doppler and residual carrier frequency offset.

The subcarrier spacing is inversely proportional to the number of subcarriers N. This means that as N increases you will get higher spectral efficiency, but at the cost of increased sensitivity to frequency offsets.

It is common for not all subcarriers to be used in an OFDM system. Some subcarriers are usually "nulled" or "zeroed out" in the frequency domain (i.e., the null tones). The zero frequency or DC is commonly nulled due to RF distortion at DC. Also, guard bands (i.e., frequencies at the edges of the frequency response corresponding to those around $s[N/2]$) are commonly nulled to prevent interference with signals in adjacent frequency bands. In the lab we assume that K subcarriers are zeroed with the zero locations being specified separately.

2 Pre-Lab

In this lab you will implement the OFDM modulator and demodulator as described in the background section. You are required to build two VIs: *OFDM_modulate.vi* and *OFDM_demodulate.vi*. These VIs will implement the appropriate transmit and receive operations from Figure 1.

Before discussing the details of these VIs, you will learn about a set of helper VIs which have been provided to you in *OFDM_comm1.0.llb*, a new digital communications library for processing related to OFDM. The VIs described in Table 1 are located in *OFDM_comm1.0.llb* and may be helpful in your implementations of OFDM modulation and demodulation.

Name of VI	Description
S2P.vi	converts serial input stream (1-D array) to parallel blocks structure (2-D array in which each row represents M consecutive elements from the input stream)
P2S.vi	converts parallel input stream (2-D array) to a serial output stream (1-D array which contains successive rows of input concatenated together)
OFDM_insert_null_tones.vi	insert columns of zeros into 2-D input array at specified locations
OFDM_add_CP.vi	prepends last C columns of 2-D input array to start of array
OFDM_remove_null_tones.vi	remove specified columns from 2-D input array
OFDM_remove_CP.vi	remove first L_c columns from 2-D input array
OFDM_FEQ.vi	computes frequency domain equalizer based on channel estimate and equalizes parallel input stream

Table 1: Helper VIs located in *OFDM_comm1.0.llb*.

To efficiently process parallel inputs, these VIs use a parallel block structure represented in a 2-D array. In this structure, each row of the 2-D array corresponds to a single block of data. Thus, the number of rows corresponds to the number of blocks that are being processed.

Note: For-Loops in LabVIEW provide an auto-indexing feature which can be used to extract the i^{th} element of a 1-D array. Auto-indexing can also be used on a 2-D array to extract the i^{th} row of the 2-D array.

Tables 2 and 3 describe the VIs that you must construct. Many of the OFDM related parameters you will need to use (such N, L_c, and *null tones*)

Table 2: Description of *OFDM_modulate.vi*

OFDM_modulate.vi - implements the OFDM modulator described in Section 1.1			
INPUTS	*input symbols*	1-D array of CDB (complex doubles)	input symbol stream to be modulated using OFDM
OUTPUTS	*output samples*	1-D array of CDB	output sample stream after OFDM modulation; one OFDM symbol corresponds to $(N + L_c)$ samples

Table 3: Description of *OFDM_demodulate.vi*

OFDM_demodulate.vi - implements the OFDM demodulator described in Section 1.2			
INPUTS	*input samples*	1-D array of CDB	input sample stream to be demodulated using OFDM; one OFDM symbol corresponds to $(N + L_c)$ samples
	channel estimate	1-D array of CDB	channel estimate computed prior to demodulation (used for FEQ)
	number of data symbols	I32	number of data symbols to be recovered[1]
	equalize channel?	Boolean	determines whether or not you should apply FEQ to OFDM demodulated symbols; set this value to TRUE by default
OUTPUTS	*demodulated symbols*	1-D array of CDB	stream of output symbols after OFDM modulation and FEQ (when appropriate)
	FD channel estimate	1-D array of CDB	frequency domain response of channel estimate computed taking DFT of channel estimate

can be found unbundled from the *OFDM parameters in* cluster. After building your VIs, insert your code into *OFDM_transmitter.vi* and *OFDM_receiver.vi* as shown in Figure 3 and 4 respectively.

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure

[1] Figure 1 implies that the output of the OFDM demodulator produces multiples of N symbols. You should truncate the array to the appropriate size (as dictated by *number of data symbols*).

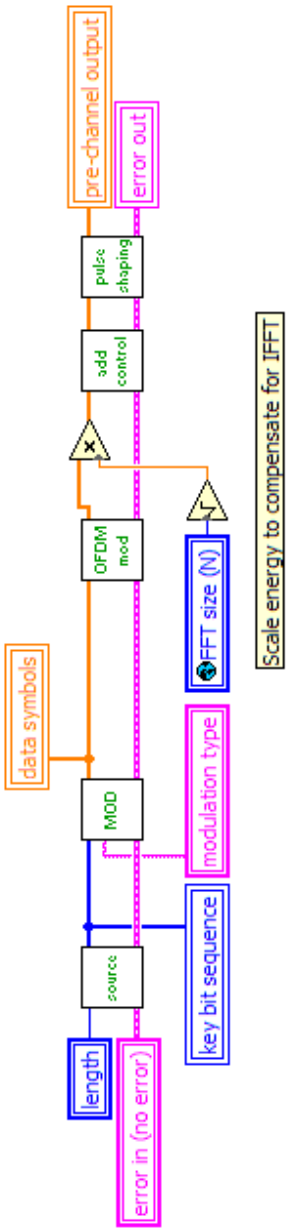

Figure 3: Block diagram for *OFDM_transmitter.vi*.

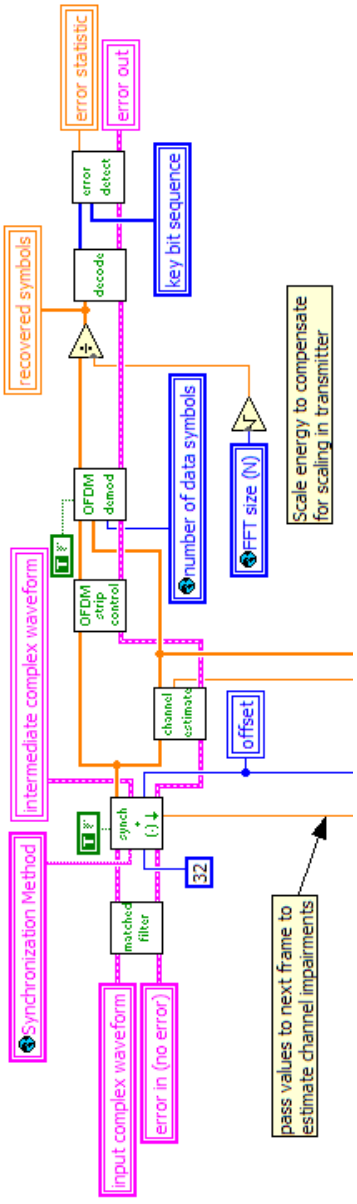

Figure 4: Block diagram for *OFDM_receiver.vi*.

that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break. In addition, you will now have access to the *OFDM parameters* cluster for OFDM-specific parameters you will need for you VIs. These are accessed in the same way that parameters from the *modulation parameters* cluster are.

After inserting your code into the simulator provided to you, verify that your code is working properly by observing that the transmitted and received constellations reflect the appropriate PSK modulation scheme you have chosen.

Pre-Lab Turn In

1. Submit your implementation of *OFDM_modulate.vi* and *OFDM_demodulate.vi*. Remember, you will be penalized if you do not wire error cluster inputs and outputs. *Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.*

2. **EXTRA CREDIT (20pts)** - implement *FEQ.vi* (i.e., your own version of *OFDM_FEQ.vi*). Table 4 describes the details of this VI. Submit your implementation of *FEQ.vi* with your other VIs. Remember to wire error clusters.

Table 4: Description of *FEQ.vi*

FEQ.vi - performs frequency domain equalization as described in Section 1.3			
INPUTS	*input*	2-D array of CDB	parallel block structure of symbols (i.e., output of DFT); each row corresponds to a block of N symbols (i.e., $\{Y[k]\}_{k=0}^{N-1}$)
	channel estimate	1-D array of CDB	channel estimate computed prior to demodulation
OUTPUTS	*equalized output*	2-D array of CDB	equalized symbols; each row corresponds to a block of N symbols (i.e., $\{\widetilde{X}[k]\}_{k=0}^{N-1}$ where $\widetilde{X}[k] = Y[k]/\widehat{H}[k]$)
	FD channel estimate	1-D array of CDB	frequency domain response of channel estimate computed by taking DFT of channel estimate (i.e., $\widehat{H}[k] = DFT\{h[l]\}$)

3 Lab Experiment

A single carrier system operating with symbol rate $1/T$ would transmit one symbol every symbol period T. Thus in time NT, the single carrier system would transmit N symbols; maintaining orthogonality (or separation) between symbols through time division multiplexing (or TDM). As you have learned in previous labs, a frequency selective channel can undo this orthogonality, making some form of equalization necessary.

Multicarrier systems use a cyclic prefix to enable digital frequency domain equalization. An interpretation is that OFDM divides a frequency selective channel into N flat fading subchannels. Thus OFDM can be seen as N parallel flat fading channels, (each of bandwidth $1/NT$), multiplexed in the frequency domain. Hence, a simple zero-forcing equalizer can be applied to each subchannel in order to equalize channel impairments. In this lab you will examine the frequency selectivity of wireless channels and explore when it might be advantageous to use OFDM to forgo the complexities of linear equalization.

Along with the advantage of low complexity equalization, OFDM systems also provide a framework which allows for many advanced digital communication techniques such as adaptive modulation and power control. Additionally, the frequency domain interpretation of OFDM (i.e., as a set of N parallel flat fading subchannels) allows for many clever techniques with regard to error control coding by taking advantage of frequency diversity (topics that are outside of the scope of this course). All of these advantages do not come for free. There are a number of tradeoffs associated with single and multicarrier systems. In this lab you will explore one particular tradeoff, the sensitivity of OFDM systems to frequency offsets.

In this lab you will run your implementation of *OFDM_modulate.vi* and *OFDM_demodulate.vi* over a real wireless link. As in previous labs, you will complete the framework for the transmitter and receiver blocks in lab using your code from this and previous labs. Insert your code into this framework as you did in the pre-lab. You will then perform two experiments to explore the previously mentioned topics: 1) the frequency selectivity of wireless channels and 2) the sensitivity of OFDM systems to frequency offsets.

3.1 Insert Your Code

Figure 5 depicts the dependencies between files in the new OFDM framework you will be using in lab. Note that this new framework still leverages many of the blocks in the original digital communications library (*digital_comm.llb*), but also includes some of the elements from the new OFDM digital communications library (*OFDM_comm1.0.llb*).

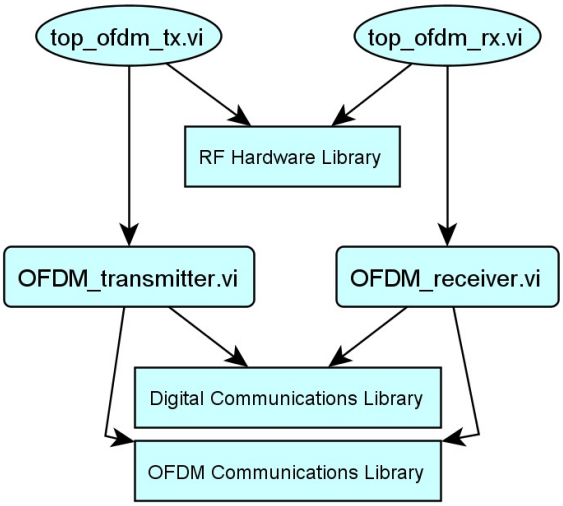

Figure 5: Hierarchy of files for OFDM framework.

Once you have inserted your code for *OFDM_modulate.vi* and *OFDM_demodulate.vi* into *OFDM_transmitter.vi* and *OFDM_receiver.vi* respectively, you will need to set up the parameters listed below. These parameters are located on the front panels of *top_tx.vi* and *top_rx.vi*. Leave any unspecified parameters set to their default value.

- Packet length = 500 bits
- Modulation type = QPSK
- Channel estimate length = 4
- FFT size (N) = 64
- Length of CP (L_c) = 8
- Null tones = {0, 31, 32, 33}

3.2 Frequency Selectivity of Wireless Channels

In this first experiment you will observe the frequency response of narrowband and wideband channels. Set up the following parameters for a narrowband system.

- TX sample rate = 4 MSamp/sec

- TX oversample factor = 20
- RX sample rate = 4 MSamp/sec
- RX oversample factor = 20
- Capture time = 2.4 msec

After transmitting a packet successfully, observe the frequency response of the narrowband channel, using the *Channel Response* graph located on the front panel of *OFDM_receiver.vi*. Also examine the instantaneous power-delay profile using the appropriate graph on the front panel of the same VI. Take note of the effective length of the channel response (i.e., the number of nonzero taps in the channel).

Next, you will observe the frequency response of a wideband channel. To set up a wideband system, set the following parameters in your OFDM system.

- TX sample rate = 20 MSamp/sec
- TX oversample factor = 4
- RX sample rate = 10 MSamp/sec
- RX oversample factor = 2
- Capture time = 100 μsec

For this part of the experiment it is critical that you place your antennas at an elevated height so that all reflected paths in the lab environment reach the antennas. After transmitting a packet successfully observe the frequency response and power-delay profile of the wideband channel in *OFDM_receiver.vi*. Also, take note of the effective length of the channel response.

Questions

Answer the following questions regarding the frequency selectivity of wireless channels.

1. According to the parameters above, what is the respective OFDM symbol rate in the narrowband and wideband systems you have set up (i.e., the reciprocal of the OFDM symbol period as previously defined)?
2. What are the effective lengths of the narrowband and wideband channels respectively?
3. Describe the frequency responses of each channel. In particular, are the frequency responses of these channels frequency selective or flat?

4. Consider the multipath channel model in the absence of noise

$$y[n] = \sum_{l=0}^{L_h} h[l]x[n-l], \tag{2}$$

where $h[n] = 0, \forall n \notin (0, 1, \ldots, L_h)$.

- Show that in an OFDM system, when $L_h = 0$ the frequency response of the channel is necessarily flat fading. In other words, show that the frequency response of all subchannels is the same (i.e., $H[n] = H[m], \forall n, m$, where $H[k] = DFT\{h[l]\}$).
- Show that when $L_h > 0$ the frequency response of the channel is frequency selective (i.e., show that $L_h > 0 \Rightarrow H[n] \neq H[m]$ for at least one different n and m).

Submit your answers to these questions as part of your lab report.

3.3 Sensitivity to Frequency Offsets

In this experiment you will observe how the performance of an OFDM system degrades in the presence of a frequency offset. Set up the following parameters in your system.

- TX sample rate = 20 MSamp/sec
- TX oversample factor = 20
- RX sample rate = 4 MSamp/sec
- RX oversample factor = 4
- Capture time = 500 μsec
- Frequency offset (Hz) = 200 Hz

To observe how a frequency offset impacts your system, you will need to disable the frequency offset correction algorithm implemented in the synchronization block. To do this, simply set *Correct Frequency Offset* on the front panel of *top_ofdm_rx.vi* to FALSE. Now you will consider two OFDM systems which use different values for N. First, consider a system which uses $N = 64$ subcarriers. Set up the following OFDM parameters.

- FFT size (N) = 64
- Length of CP (L_c) = 8

- Null tones = {0, 31, 32, 33}

Observe how a frequency offset impacts the received signal constellation. Also make note of the impact of a frequency offset on BER performance. Increase the amount of frequency offset in your system in increments of 200 Hz and observe how the signal constellation and BER performance vary. Now consider a system which uses $N = 1024$ subcarriers.

- FFT size (N) = 1024
- Length of CP (L_c) = 32
- Null tones = {0, 511, 512, 513}

Again, observe how a frequency offset of just 200 Hz impacts the received constellation and BER performance of your system. To understand how a frequency offset impacts OFDM systems, it is convenient to think of a frequency offset as a shift in the frequency domain. This shifting can cause what is known as inter-carrier interference (or ICI).

Questions

Answer the following questions about the sensitivity of OFDM systems to frequency offsets.

1. Recall from the Lab 5 that in a single carrier system, a frequency offset causes a time varying phase offset which effectively "smears" the received constellation as it rotates it. How is the impact of a frequency offset in OFDM systems different from that of single carrier systems? In particular, how is the impact on the signal constellation different?
2. What is the subcarrier spacing Δ_c of your system when $N = 1024$ and 64 respectively?
3. Which of the systems (i.e., $N = 1024$ or 64) is more severely impacted by a 200 Hz frequency offset? Why?
4. As mentioned, frequency offsets can be interpreted as shifts in the frequency domain. There is a duality between the effect of this shifting in a multicarrier system and symbol timing error in a single carrier system (discussed in Part 2 of Lab 2).

   Discuss this relationship between the effect of symbol timing error in single carrier systems and frequency offset in OFDM systems. Be sure to include in your discussion the effect of each impairment on the received signal constellation. *Hint: Think about the impact of each type of error on the orthogonality (or separation) between symbols.*

Submit your answers to these questions as part of your lab report.

3.4 Lab Turn In

Show the instructor that you have replaced the OFDM modulator and demodulator in lab with your own code. Demonstrate to the instructor that your code is working properly. Also show the instructor that your OFDM system "breaks" down when you have frequency offsets greater than the subcarrier spacing that are not corrected for (i.e., when $f_{\text{offset}} > \Delta_c$). Answer all of the questions above and submit your answers in your lab report.

DIRECTIONS FOR LAB SUBMISSION

Your lab report should include the following.

1. Answer all questions from the lab experiment (i.e., do not reanswer pre-lab questions).

2. Discuss any problems you might have encountered and how you overcame these obstacles.

3. In this lab you observed how an OFDM system operates over a real wireless link. Answer the following questions about your experience in lab.

   (a) OFDM systems have an inherent overhead due to the need for a cyclic prefix. Assuming QPSK modulation, in terms of N, L_c, T, and the number of null tones K (all as previously defined), what is the effective data rate of the OFDM system in lab?

   (b) Based on your experience in lab, discuss why or why not you might use an OFDM system in a wideband and narrowband system respectively. Include in your discussion some of the benefits/costs of each scenario.

   (c) OFDM systems in general have a large degree of flexibility. Name at least three parameters of OFDM systems which contribute to this flexibility and comment on how they do so.

Lab 7: Synchronization in OFDM Systems using Schmidl and Cox Algorithm

Summary

In Lab 6 you implemented key features of the orthogonal frequency division multiplexing (OFDM) multicarrier modulation technique. In particular you implemented the encoding from frequency domain to time domain at the transmitter, and the low complexity frequency domain equalization at the receiver. Systems that employ OFDM modulation, like those that employ single carrier modulation, also require frame synchronization and carrier synchronization, not to mention channel estimation. In Lab 6, frame and carrier synchronization was performed using the single carrier modulated training sequence from previous labs. In this lab you will perform frame and frequency synchronization using an OFDM modulated training sequence, using the short training sequence, as defined by the IEEE 802.11a standard, to perform synchronization. The OFDM system considered in this lab is illustrated in Figure 1.

The emphasis of this lab is using an OFDM modulated training signal to perform carrier and frequency synchronization. First you will review the IEEE 802.11a preamble including short and long training sequences. Then you will learn about the Schmidl and Cox Algorithm (SCA), which leverages the periodic structure of the OFDM training sequence to perform frame and frequency synchronization.

For the pre-lab submission, you have to turn in the VI described in Section 2 (*SchmidlCox.vi*). Further, you have to submit the answers to questions in Section 2.

1 Background

1.1 Impairments in Frequency Selective Channel

In this section we review the baseband model for delay and frequency offset in the presence of a multipath channel. Referring to Figure 1, if the carrier offset is small and sufficient sampling is employed, then the discrete-time signal at the receiver after matched filtering and downsampling can be written as

$$y[n] \quad = \quad e^{j2\pi\epsilon n} \sum_{l=0}^{L} h[l]w[n-l-D] + v[n],$$

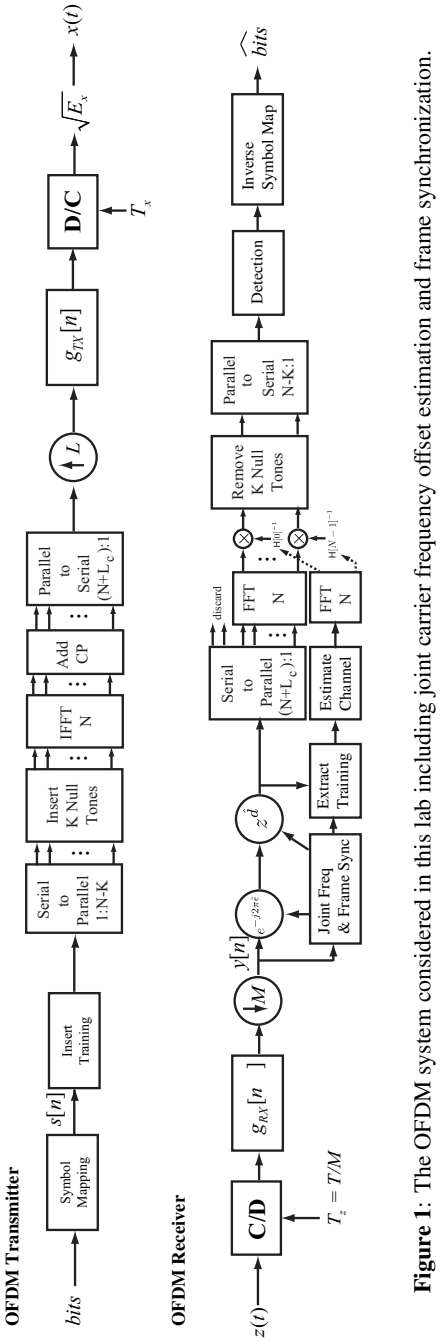

Figure 1: The OFDM system considered in this lab including joint carrier frequency offset estimation and frame synchronization.

where ϵ is the discrete-time frequency offset, D is an unknown delay, and $\{h[\ell]\}_{\ell=0}^{L_h}$ are the coefficients of the unknown frequency selective channel. The frequency offset is estimated and removed during carrier frequency offset synchronization. The delay D is estimated and removed as part of frame synchronization. The channel coefficients are estimated and used to design an equalizer.

1.2 IEEE 802.11a Short Training Sequence

In this lab you will use a carefully constructed training sequence to perform frequency offset synchronization and frame synchronization. Figure 2 shows the IEEE 802.11a training structure described in the Physical Layer Specification [4]. The training sequence is composed of a "short" and "long" training sequence. The short training sequence is composed of 10 identical subsequences and has a total length of $N_t = 160$. The long training sequence is composed of two identical subsequences with an extra long cyclic prefix. In this lab you will use the short training sequence. The frame structure will be the same as the Lab 6 with the time domain training replaced by the short training sequence.

The short training sequence is created from two OFDM symbols in a very special way. We describe how it is performed mathematically here. Consider the symbols $\{s[n]\}_{n=0}^{N-1}$ that will be used to generate the coefficients for one OFDM symbol. Suppose that $s[n]$ is generated by upsampling the symbols $\{\bar{s}[n]\}_{n=0}^{N/L}$ where L is an integer. First consider the IDFT with zero cyclic prefix. Then

$$w[n] = \frac{1}{N}\sum_{m=0}^{N-1} s[m]e^{j\frac{2\pi mn}{N}}. \quad (1)$$

Remembering that $s[Ln] = \bar{s}[n]$ and that $s[n] = 0$ for values of n that are not an integer multiple of L, it follows that

$$\begin{aligned} w[n] &= \frac{1}{N}\sum_{m=0}^{N/L-1} s[Lm]e^{j\frac{2\pi Lmn}{N}} \\ &= \frac{1}{N}\sum_{m=0}^{N/L-1} \bar{s}[m]e^{j\frac{2\pi mn}{N/L}}. \end{aligned}$$

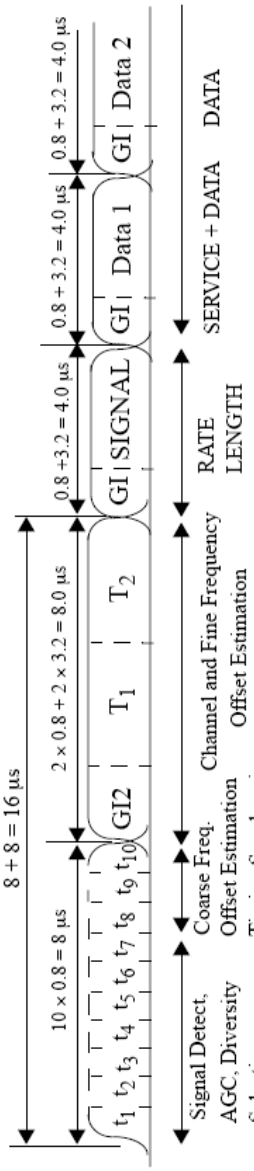

Figure 2: IEEE 802.11a training structure.

Because of the periodicity of the complex exponential, it can be verified that for $n = 0, 1, \ldots, N/L - 1$

$$\begin{aligned} w[n + kN/L] &= \frac{1}{N} \sum_{m=0}^{N/L-1} \bar{s}[m] e^{j \frac{2\pi m(n+kN/L)}{N/L}} \\ &= \frac{1}{N} \sum_{m=0}^{N/L-1} \bar{s}[m] e^{j \frac{2\pi mn}{N/L}} e^{j \frac{2\pi mkN/L}{N/L}} \\ &= \frac{1}{N} \sum_{m=0}^{N/L-1} \bar{s}[m] e^{j \frac{2\pi mn}{N/L}} \\ &= w[n]. \end{aligned}$$

This means that by zero every $L - 1$ subcarriers, it is possible to make the OFDM symbol look periodic with L periods.

To conclude the discussion now suppose there is a cyclic prefix of length L_c and that $N/L = L_c$

$$w[n] = \frac{1}{N} \sum_{m=0}^{N-1} s[m] e^{j2\pi \frac{m(n-L_c)}{N}} n = 0, ..., N + L_c - 1. \qquad (2)$$

Remembering the zero subcarriers for $n = 0, 1, \ldots, N + L_c - 1$

$$\begin{aligned} w[n] &= \frac{1}{N} \sum_{m=0}^{N/L-1} \bar{s}[m] e^{j2\pi \frac{m(n-L_c)}{N/L}} \\ &= \frac{1}{N} \sum_{m=0}^{L_c-1} \bar{s}[m] e^{j2\pi \frac{m(n-L_c)}{L_c}} \\ &= \frac{1}{N} \sum_{m=0}^{L_c-1} \bar{s}[m] e^{j2\pi \frac{mn}{L_c}}, \end{aligned}$$

which satisfies the property that $w[n+kL_c] = w[n]$ for $n = 0, 1, \ldots, L_c - 1$ and $k = 0, 1, \ldots, N/L_c$. Therefore the cyclic prefix becomes an additional period of repetition!

The IEEE 802.11a standard [4] specifies that $N = 64$ and $L = 4$ in the short training sequence, thus yielding a training sequence which is periodic with period $L_c = 16$, which is also the length of the cyclic prefix. Two OFDM symbols concatenated together form a sequence with 160 samples, that consists of 10 repetitions of the same sequence. You will use this training sequence in the lab.

1.3 The Schmidl and Cox Algorithm

The Schmidl and Cox Algorithm (SCA) [16] is a clever specialization of the Moose algorithm [7] to OFDM waveforms. It uses periodicity created in the OFDM symbol to perform the frequency offset correction and frame synchronization. A main difference over the Moose algorithm is that it consists of two steps of frequency offset correction. The first step corrects for offsets that are a fraction of $1/N$, the subcarrier spacing. The second step corrects for integer offsets that are multiples of the subcarrier spacing. The integer offset is performed using a specially constructed second training symbol. In this lab we focus on the fine frequency offset correction.

The training sequence considered in the lab has 10 repetitions of the same sequence in the 160 samples. For maximum compatibility and reuse with the Moose algorithm developed in previous labs, we treat this sequence as two repetitions of length $W = 80$. There are algorithms that can exploit the multiple periods of repetition, however, they will not be considered in this lab.

The steps of the SCA that you will implement are summarized as follows.

1. The first step is to compute the correlation metric

$$R[d] \quad = \quad \sum_{n=0}^{N-1} y^*[n+d]y[n+d+W], \tag{3}$$

   where W is half the size of the periodic training sequence and N is the number of subcarriers.

2. The second step is to compute the average received signal power

$$P[d] \quad = \quad \sum_{n=0}^{N-1} |y[n+d+W]|^2. \tag{4}$$

3. From $P[d]$ and $R[d]$, compute $M[d] = \left| \dfrac{R[d]}{P[d]} \right|^2$.

4. Approximate delay parameter D by solving for $\hat{d} = \underset{d}{\text{argmax}}\, M[d]$ searching over a reasonable range of samples.

5. Estimate the frequency offset parameter as

$$\hat{\epsilon} = \text{phase} \frac{P[D]}{R[D]}. \tag{5}$$

6. Correct for the frame offset and frequency offset as in the Moose algorithm.

The algorithm described above is an abbreviated version of the full SCA discussed in [16]. The full version of this algorithm includes a method for coarse frequency offset estimation/correction[1], which will not be discussed in this lab.

2 Pre-Lab

Questions

Answer the following questions before implementing the Schmidl and Cox algorithm.

1. In SCA, the delay estimate $\hat{d}$ is computed through an exhaustive search over the set of valid delay values (i.e., the search window). Suppose that the length of the captured sequence is C samples. In terms of C, N, L_c, and N_{ofdm} (the number of OFDM symbols in the OFDM payload, which is described in Lab 6), what is the set of valid delay values you should search through to find $\hat{d}$?

2. For the IEEE 802.11a short training sequence being used in lab, what is the range of frequency offsets that can be estimated/corrected using SCA (or equivalently, the Moose algorithm)? You may assume an OFDM sample rate of $1/T = 1MHz$.

3. Given that the IEEE 802.11a short training sequence contains 10 periods of a length 16 sequence (as described above), propose a frequency offset scheme that allows you to correct for a greater range of frequencies than computed in the last question. You may assume that the frame detection problem has been solved, and that an aligned sequence is available.

Programming

In this lab you will be implementing the Schmidl and Cox algorithm for frame and frequency synchronization. You may use your previously constructed implementation of the Moose algorithm (or the one available in the digital communications library provided to you) for frequency offset estimation/correction. This means that you will only need to implement the steps

[1] Coarse frequency offset is the same as a modulo offset [i.e., an integer multiple of $1/(2\pi WT)$].

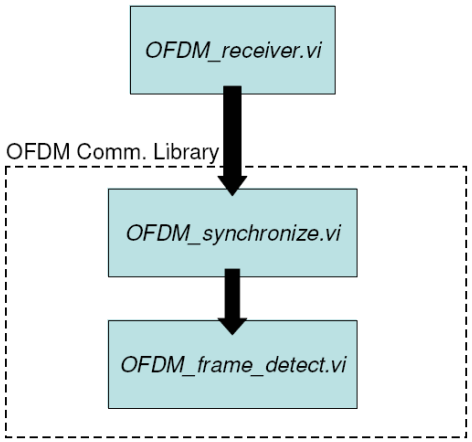

Figure 3: Hierarchy of files associated with *OFDM_frame_detect.vi*.

of SCA that pertain to frame detection (as described in Section 1.3). You will build the VI *SchmidlCox.vi*, described below. After building this VI you will insert it into *OFDM_frame_detect.vi* shown in Figure 4. You can find *OFDM_frame_detect.vi* in the block diagram of *OFDM_synchronize.vi*. The hierarchy of these files is shown in Figure 3.

As stated in Table 1 and mentioned above, you do not need to reimplement the Moose algorithm for frequency offset estimation/correction. Reuse your previous implementation of this algorithm or the one provided to you in the digital communication library.

You have been provided with templates for the VIs you need to create for this lab that already have all the inputs and outputs wired for you. What is required of you is to finish constructing the block diagram to provide the functionality of the VIs.

Throughout the course, several of the VIs you will create will have a *modulation parameters* cluster passed into and out of them. The *modulation parameters in* contains many of the parameters needed by your VIs and will be unbundled from them. This has already been done for you in the template VIs if necessary. Some VIs will also have *modulation parameters out*, so that the cluster can be passed on through to the VIs that follow. Please ensure that these clusters remain wired the way they are in the template VIs, as changing how they are wired will cause VIs further down the line to break. In addition, you will now have access to the *OFDM parameters* cluster for OFDM-specific parameters you will need for you VIs. These are accessed in the same way that parameters from the *modulation parameters* cluster are.

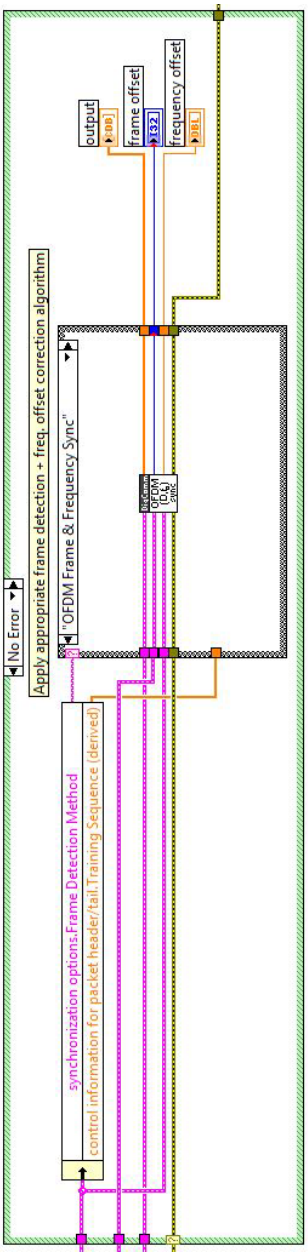

Figure 4: Block diagram of *OFDM_frame_detect.vi*.

Table 1: Description of *SchmidlCox.vi*

SchmidlCox.vi - implements SCA for frame and frequency synchronization; uses Moose algorithm to perform frequency synchronization.			
INPUTS	*input complex waveform*	IQ waveform cluster	received sequence after matched filter and symbol timing recovery
	correct frequency offset	Boolean	determines whether or not your VI should correct for the frequency offset estimated by *moose.vi*
OUTPUTS	*output*	1-D array of CDB	received sequence after correcting for estimated delay $\hat{d}$ and frequency offset (when appropriate)
	frame offset	I32	estimated delay $\hat{d}$ computed by SCA
	frequency offset	DBL	frequency offset estimate computed by Moose algorithm

After inserting your code into the simulator, verify that it correctly finds the start of the frame and corrects for frequency offset. You should test that your code works for nontrivial delay and frequency offset up to f_M.

Note: The suboptimal Schmidl and Cox algorithm presented here does not work in the presence of noise or multipath; do not attempt to optimize the algorithm to compensate for this shortcoming.

You will need to submit a plot (or screen capture of a plot[2]) of $M[d]$ the frame correlation metric. Generate the plot using $-\infty$ dB of noise power and a delay of 80 μsec. Please leave all other parameters set to their default value.

Pre-Lab Turn In

1. Submit your implementation of *SchmidlCox.vi*. Make sure you use the parameters *Training Sequence FFT Size* and *Training Sequence Length of CP* from the *OFDM parameters in* cluster in your implementation, as these are parameters corresponding to the training sequence and not the OFDM payload.

[2]You can plot a 1-D array of *DBL* in LabVIEW, simply by placing a waveform graph on the front panel of your VI and connecting its terminal to the appropriate array on the block diagram of your VI.

Remember, you will be penalized if you do not wire error cluster inputs and outputs.

Note: If you need to generate additional subVIs for this or any future labs, please submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section.

3. Submit your plot of $M[d]$; generated using the parameters listed below.

   - Noise power = $-\infty$ dB
   - Delay = 80 μsec

3 Lab Experiment

In the pre-lab you implemented the Schmidl and Cox algorithm and observed its performance in channels with delay and frequency offset. In this lab experiment you will learn about how noise impacts SCA and an optimization to SCA which increases the robustness of this frame and frequency synchronization method for OFDM systems.

3.1 Impact of Noise on Schmidl and Cox Algorithm

As discussed in Section 1.3, SCA attempts to maximize the correlation between the first and second halves of the training sequence. Note that the training sequence is composed of two identical OFDM symbols [including their respective cyclic prefixes (i.e., $N_t = 2(N + L_c)$)]. Because of the cyclic prefix (or CP), the objective function[3] $M[d]$ is the same for all delay values $d \in (D + L_h, \ldots, D + L_c)$ in the absence of noise, where D is the actual delay of the channel. This phenomenon can be graphically interpreted as a plateau of length $L_c - L_h$, i.e., on which $M[d]$ is maximized.

When noise is added to the system, the entire objective function is perturbed as shown in Figure 5. The once flat plateau of $M[d]$ has now been distorted. Thus, finding the argument which maximizes $M[d]$ may give only a rough, often erroneous, approximation of the delay (or frame offset).

Notice that qualitatively speaking the perturbed objective function $M[d]$ still retains an approximate plateau. Intuitively, this suggests that a better approximation of delay might be found by finding the "start" of the plateau. In the following sections, you will learn more about frame detection errors and a formal definition of the intuitive method alluded to.

[3]The term *objective function* refers to the function being optimized in a given optimization problem.

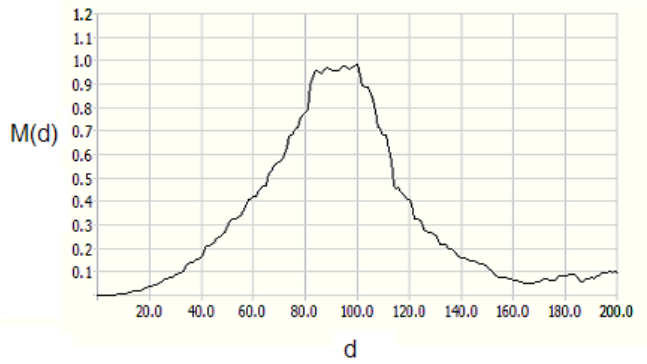

Figure 5: Objective function $M[d]$ from SCA. Noise power $N_o = -20$ dB and delay $D = 80$.

3.2 Good and Bad Frame Synchronization Error in OFDM Systems

In Lab 6 that dealt with frame and frequency synchronization you learned how a frame detection error can lead to severe bit error rates (i.e., a frame offset estimation error of one can lead to a BER of 0.5). To understand why this is not necessarily the case in OFDM systems, consider the impact of cyclic shifts on the DFT of a periodic discrete sequence. Let $x[n]$ be a periodic sequence with period N and $X[k]$ be the discrete Fourier transform of $x[n]$. If $z[n] = x[(n - S)$ modulo $N]$ (i.e., $x[n]$ after a cyclic shift of S), then

$$Z[k] = \sum_{n=0}^{N-1} z[n] e^{-j2\pi \frac{kn}{N}} \tag{6}$$

$$= X[k] e^{j2\pi \frac{kS}{N}}, \tag{7}$$

describes the DFT of the shifted sequence $z[n]$. This implies that a cyclic shift of S imparts a $\frac{kS}{N}$ phase offset to the k^{th} subcarrier. Therefore, in an OFDM system, any frame detection error which leads to a cyclic shift of the OFDM samples (i.e., after removing the cyclic prefix) will only impart a constant phase offset to each subcarrier. This phase offset can be accounted for by a properly formulated channel estimation algorithm.

Consider a frame detection error $\nu = \hat{d} - D$, where D is the actual delay of the channel. After correcting for the estimated offset $\hat{d}$, the resulting sequence may not be aligned to the boundary of an OFDM symbol period. This means that the start of the first OFDM symbol (i.e., the N samples not including the CP) is located at $L_c + \nu$.

In the OFDM demodulation algorithm, the first L_c samples of each OFDM symbol are removed before demodulation by a DFT. Thus, if $-(L_c - L_h) \leq \nu < 0$, then the OFDM samples of each symbol undergo a cyclic shift of $|\nu|$ (after removing the CP). As discussed earlier, such a frame detection error leads to phase shifts in the DFT of each OFDM symbol, which can be accounted for by an appropriate least squares channel estimator. Thus this type of frame detection error is considered to be acceptable.

Consider the scenario in which $\nu > 0$. In this situation, each OFDM symbol (after removing the CP) contains $|\nu|$ samples from an adjacent symbol. This means that every subchannel will be corrupted by data from an adjacent OFDM symbol; this is an impairment which cannot be undone by a simple frequency domain equalizer (FEQ). It is also interesting to note that this type of impairment must be modeled as a non-causal system. For these reasons, a frame detection error $\nu > 0$ is not considered acceptable (i.e., a "bad" error).

In the next section you will learn about an optimization of SCA that tends to produce a frame detection error $\nu < 0$ (i.e., typically underestimates delay).

Questions

Include the answers to the following questions in your lab report.

1. Prove that the DFT of a cyclically shifted periodic sequence results in a frequency dependent phase shift [i.e., given $x[n]$ and S, as defined above, prove that Eq. (7) is true].

2. Using the simulator provided to you (i.e., without inserting your own code), you will observe how overestimating and underestimating delay impacts bit error rate. Set the noise power $N_o = -10$ dB. Describe what happens to the average bit error rate of the system when the estimated delay exceeds the actual delay of the channel. You may have to run the simulator multiple times before such a frame detection error occurs.

3. Based on the discussion in this section and your simulation experience, is it better to underestimate or overestimate the delay of the channel? Justify your answer.

3.3 Optimizing the Schmidl and Cox Algorithm

The Schmidl and Cox algorithm, as described in Section 1.3, does not perform well in channels with even a moderate amount of noise. This poor performance stems from the fact that SCA typically overestimates the delay of the channel. You can use your simulator to verify this behavior.

In [16] an optimization for SCA is proposed. You will implement a slight variation of this optimization. Recall that even in the presence of noise, $M[d]$ has a plateau of approximate length $L_c - L_h$. This optimization seeks to estimate the beginning of that plateau. The algorithm you will be implementing is described by the steps below.

1. Given received sequence $y[n]$, compute $M[d]$ as described in SCA (see Section 1.2).

2. Find $M_{max} = \max_d M[d]$.

3. Let $\mathcal{D} = \{d | M[d] \geq 0.90(M_{max})\}$.

4. Estimate delay by solving for $\hat{d} = \inf \mathcal{D}$.

5. After correcting for offset $\hat{d}$, apply Moose algorithm for frequency offset correction.

This algorithm will be referred to as optimized SCA. The set $\mathcal{D}$ contains an optimistic[4] estimate of the possible delay values which make up the plateau. By taking the infimum of this set, optimized SCA estimates a lower bound on the delay of the channel.

After applying this optimization to your implementation of *SchmidlCox.vi*, you will test your code over a real wireless link. Insert your code for optimized SCA into the framework for the receiver in lab. Leave all parameters in your system set to their default values.

Questions

Include the answers to the following questions in your lab report.

1. Graph $M[d]$, a 1-D array of CDB, on the front panel of *SchmidlCox.vi* using a waveform graph. Discuss the relationship between the estimated frame offset computed by optimized SCA and the beginning of the $M[d]$ plateau (which you should be able to approximate from your graph). Please include a screen capture of your graph of $M[d]$ in your lab report and refer to it in your discussion.

2. What assumption has been made regarding the quantity $L_c - L_h$? Why is this assumption critical?

[4]The set $\mathcal{D}$ is said to be optimistic because the set forms a superset of the values that actually make up the length $L_c - L_h$ plateau of $M[d]$.

3.4 Lab Turn In

Demonstrate to the instructor that your code is working properly. Show the instructor that your implementation of optimized SCA still "breaks" when the algorithm overestimates the delay of the channel. Answer the questions below.

Additional Questions

In this lab you learned about optimized SCA and how it operates over a real wireless link. Answer the following questions about your experience in lab.

1. Compare and contrast the following synchronization methods you have learned about in lab: (1) the self-referenced frame detection method for a single carrier training sequence[5] and (2) optimized SCA for an OFDM training sequence[6].

2. Using the standard Moose algorithm, with no optimizations or modifications and the entire IEEE 802.11a short training sequence [i.e., where $(N_t = 160) \Rightarrow (W = 80)$], what is the range of correctable offsets for IEEE 802.11a systems? Is this the best that you could do?

   Hint: You may need to look at the IEEE 802.11a standard to find out the OFDM sample rate or equivalently the system bandwidth.

DIRECTIONS FOR LAB SUBMISSION

Your lab report should accomplish the following tasks.

1. Answer all questions from Lab 7 (not including pre-lab questions).

2. Discuss any problems you might have encountered in Lab 7 and how you overcame these obstacles.

[5] Here *single carrier training sequence* refers to the sequence derived from the length 11 Barker code.

[6] *OFDM training sequence* in this context refers to the IEEE 802.11a short training sequence used in lab.

Lab 8: Channel Coding in OFDM Systems

Summary

As you have learned from previous labs, one of the main advantages of OFDM is the ability to forgo the complicated equalizer design[1] needed by single carrier systems to combat intersymbol interference (ISI). For OFDM systems, to fully utilize their potential for high-data-rate communication in frequency selective (ISI) channels they must take advantage of the diversity inherent to frequency selective channels, making use of channel coding and interleaving.

In this lab you will explore the topic of channel coding. Figure 1 shows how channel coding fits into the system model for our wireless communication system. By adding structured redundancy to the transmitted sequence, channel coding provides added resilience to channel impairments and improves overall throughput. This lab begins by introducing the common classes of channel codes. You will then learn about a simple code, the repetition code, which will reveal some of the basic benefits and costs of channel coding.

The repetition encoding and decoding blocks will be provided to you. All that is required from you this lab is to compare the bit-error performance of the OFDM system you have been developing with and without coding. You will implement the code over a real wireless link to estimate coding gain (a metric of performance gain between coded and uncoded systems).

1 Background

1.1 Structured Channel Codes

Coding theory is a field which encompasses both source and channel coding. It is largely based on results from finite field algebra. The main idea behind channel coding is that adding structured redundancy to the source bits for a transmitted sequence can improve the resilience of transmission to channel distortion. In other words, a properly designed channel code can improve the bit-error rate (BER) performance of a system (i.e., reducing the transmit power required to achieve a particular error performance). A rate (k/n) code takes a length L bit sequence and generates $\frac{nL}{k}$ bits. There are two main classes of channel codes: (1) block codes and (2) convolutional codes. Channel coding is sometimes referred to as forward error correction (FEC) or error control coding, or just "coding".

[1] Through the use of guard intervals and an appropriately designed FEQ, OFDM systems can forgo the complicated equalizer design needed by wideband single carrier systems.

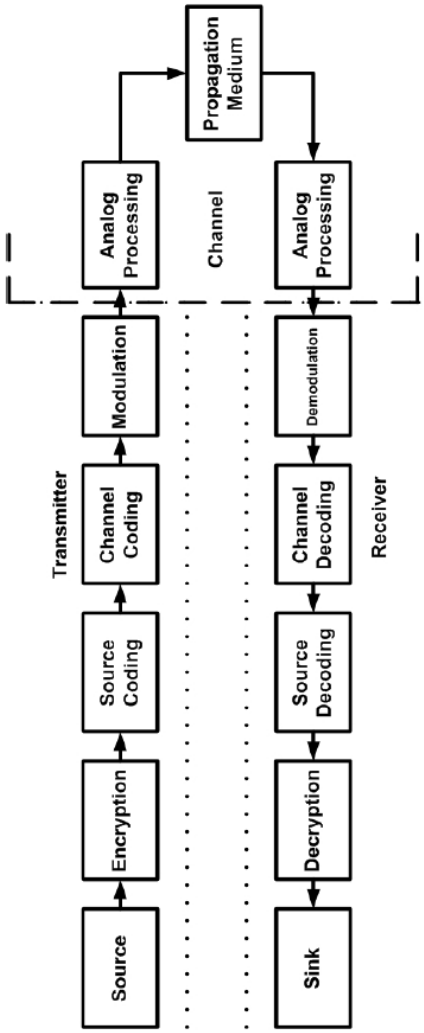

Figure 1: A typical digital communication system model. The channel coding is applied before the demodulation and the channel decoding is applied after the demodulation. In more sophisticated systems, the modulation and coding are jointly designed and decoded.

Channel Code	Code Type	Technology/Standard
Hamming Code	Block	–
Reed–Solomon	Block	disk drives, CD's, DVD's, WLAN
Punctured Codes	Convolutional	Hybrid-ARQ, IEEE 802.11a
Turbo Codes	Convolutional	3G mobile, satellite comm.
Low Density Parity Check (LDPC)	Block	–

Table 1: Various Channel Codes.

In block coding, the encoder first segments the source sequence into blocks of k bits. Thus each block represents any one of 2^k distinct messages. This block is in turn mapped to a length n bit sequence called a codeword. Each codeword contains $r = n - k$ redundant bits. This kind of code is called an (n, k) block code and can be used to detect and possibly correct errors. The resilience of block codes to channel distortion largely depends on the structure of the r redundant bits.

Convolutional codes are a class of channel codes which employ convolutional encoders. The encoded sequence is a convolution of the algebraic polynomial describing the source bit sequence with a polynomial which describes the encoder. Equivalently, the i^{th} codeword generated by a convolutional encoder is a weighted sum of the previous input messages to the encoder. This is similar to the convolution used in linear time-invariant systems and shares many of the same properties.

Convolutional codes generally have very efficient implementations which employ shift registers (to retain state or memory) and exclusive-or (XOR) gates. Block codes, which generally require more logic to implement, are much easier to analyze. There are a number of tradeoffs between both classes of codes, but neither class of codes fundamentally provides better error protection than the other. Table 1 lists a variety of new and old channel codes.

1.2 Repetition Codes

Repetition codes are a very simple type of block code. In the remainder of this section you will learn about the encoder, decoder, and probability of error analysis for a rate $\left(\frac{1}{n}\right)$ repetition code.

Encoder

A rate $\left(\frac{1}{n}\right)$ repetition code converts each message bit into a length n codeword. More precisely, let $\mathcal{I}(x)$ be a mapping from the space of message bits to length

n codewords such that

$$\mathcal{I}(x) = \begin{cases} (0, 0, \ldots, 0), & x = 0, \\ (1, 1, \ldots, 1), & x = 1. \end{cases} \tag{1}$$

Thus a rate $\left(\frac{1}{3}\right)$ repetition code has the following input-output relationship

$$\begin{aligned} 0 &\Rightarrow (0, 0, 0), \\ 1 &\Rightarrow (1, 1, 1). \end{aligned}$$

Decoder

A simple (although not necessarily optimal) decoder for a rate $\left(\frac{1}{n}\right)$ repetition code is a majority decoder. For each length n codeword received (i.e., after demodulation), the majority decoder estimates the transmitted bit sequence by finding the majority consensus of the n received bits. For example, if $n = 5$ and the receiver estimates the bit sequence 11001 after demodulation, then the majority decoder guesses that the message bit $m = 1$ was transmitted. Typically n is chosen to be odd to avoid a race condition.

Probability of Error Analysis

Suppose that after modulation, bit errors at the receiver can be modeled as an i.i.d. Bernoulli random process with parameter p, where p reflects the average bit error rate of the uncoded system. This means that a bit error occurs in the coded system only when $\lceil n/2 \rceil$ or more errors occur in a codeword. Therefore, probability of bit error is given by

$$P_e = \sum_{k=\lceil n/2 \rceil}^{n} \binom{n}{k} p^k (1-p)^{n-k}. \tag{2}$$

Note that the number of errors in each codeword is being modeled as a binomial(p) random variable. Also note that p, the probability of bit error in the uncoded system, is actually a function of SNR [i.e., $p \rightarrow p(\gamma)$, where γ represents SNR].

2 Pre-Lab

In this lab you will be implementing the encoder and majority decoder for a rate $\left(\frac{1}{3}\right)$ repetition code. Tables 2 and 3 describe the two VIs that you are required to build. You will insert your code into the baseband simulator provided to you in order to verify functionality and correctness.

Table 2: Description of *repeat_encoder.vi*

repeat_encoder.vi - implements rate (1/3) repetition code			
INPUTS	*information bit sequence*	1-D array of U8	bit sequence to be repetition encoded
	enable channel coding	Boolean	determines whether or not your VI should be using repetition code
OUTPUTS	*encoded bit sequence*	1-D array of U8	bit sequence after repetition encoding

Table 3: Description of *repeat_decoder.vi*

repeat_decoder.vi - implements majority decoding to decode rate (1/3) repetition code			
INPUTS	*estimated bit sequence*	1-D array of U8	estimated bit sequence (i.e., after demodulation)
	enable channel coding	Boolean	determines whether or not your VI should be using repetition code
OUTPUTS	*decoded bit sequence*	1-D array of U8	bit sequence after majority decoding

You will have to insert the *repeat_encoder.vi* and *repeat_decoder.vi* into the block diagram for *OFDM_transmitter.vi* and *OFDM_receiver.vi* respectively.

Once you have verified that your code is working correctly, you will then need to set up the following parameters in order to complete the remainder of the pre-lab. These parameters may be located on the front panel of *simulator.vi*.

- *Correct frequency offset?* = FALSE
- *Equalize channel?* = FALSE
- Synchronization method = Fixed Offset
- Channel model = AWGN

Leave all unspecified parameters set to their default values. Using this prescribed setup, you will compare the performance of a coded and uncoded OFDM system over an AWGN channel. You will be required to plot BER as a function of SNR for both the coded and uncoded systems.

Questions

Answer the following questions about channel coding.

1. According to the union bound for probability of error in QPSK systems, what SNR is required to maintain a symbol error rate of 10^{-3} in an uncoded OFDM system operating in an AWGN channel?

2. Consider an OFDM system with the following parameters:

   - $(N - K)$ subchannels carry QPSK modulated data,
   - a length L_c guard interval (or cyclic prefix) is used,
   - a rate $\frac{k}{n}$ channel code is used,
   - and $1/T$ is the OFDM sample rate.

   In terms of these parameters, what is the effective data rate of this coded OFDM (COFDM) system? You may ignore the overhead cost of training in your answer.

3. On the same plot, graph BER as a function of SNR for the coded and uncoded OFDM systems in an AWGN channel. Vary SNR from 0 to 10 dB in increments of 2 dB (i.e., vary noise power N_o from 0 down to -10 dB). Collect average BER statistics using the simulator with your code for repetition coding.

   You must use your LabVIEW code to generate the data for your plots. Use a logarithmic scale for BER and a linear scale for SNR in your plot.

4. From the plot you created in the last question, what is the approximate difference between the SNR required to obtain an error rate of 10^{-3} in the coded and uncoded systems?

Pre-Lab Turn In

1. Submit your implementations of *repeat_encoder.vi* and *repeat_decoder.vi* electronically. Remember to include error cluster inputs and outputs.

   Note: If you need to generate additional subVIs for this lab, please electronically submit these subVIs along with your other pre-lab VIs.

2. Submit your answers to all of the questions in the pre-lab section.
3. Submit your plot of the BER curves comparing coded and uncoded OFDM.

3 Lab Experiment

In this lab, you will evaluate the bit-error rate performance of a coded OFDM system and show that it is better than that of an uncoded system. The "gap" in the SNR required to achieve a particular bit-error probability between the coded and uncoded OFDM systems is defined as the coding gain of the channel code. In this lab you will empirically approximate the coding gain of the rate $\left(\frac{1}{3}\right)$ repetition code over a real wireless link.

3.1 Coding Gain

Coding gain is a well-established metric for gauging the performance of channel codes. To formally define coding gain, let $P_{\mathrm{e,c}}(\gamma)$ and $P_{\mathrm{e,uc}}(\gamma)$ be functions of SNR γ which describe the probability of error performance of a coded and uncoded system respectively. If we assume that both BER curves are continuously differentiable and monotonically decreasing, then there exist two functions $P_{\mathrm{e,c}}^{-1}(p)$ and $P_{\mathrm{e,uc}}^{-1}(p)$ such that $P_{\mathrm{e,c}}(P_{\mathrm{e,c}}^{-1}(p)) = p$ and $P_{\mathrm{e,uc}}(P_{\mathrm{e,uc}}^{-1}(p)) = p$ (i.e., there exists an inverse mapping from the probability of error values to SNR values for each curve). Thus, let the gap function $G(\mu)$ be defined as

$$G(\mu) \quad = \quad \frac{\mu}{P_{\mathrm{e,c}}^{-1}\left(P_{\mathrm{e,uc}}(\mu)\right)}. \tag{3}$$

The gap function $G(\mu)$ describes the gap between the coded and uncoded error curves. More precisely, it describes the ratio between the SNR required to sustain $P_{\mathrm{e,uc}}(\mu)$ error rate in the uncoded and coded systems. Thus, coding gain can be formally defined as

$$C = \lim_{\mu \to \infty} G(\mu), \tag{4}$$

or in a decibel scale

$$C_{\mathrm{dB}} = \lim_{\mu \to \infty} \left[10 \log(\mu) - 10 \log(P_{\mathrm{e,c}}^{-1}(P_{\mathrm{e,uc}}(\mu))) \right]. \tag{5}$$

Coding gain approaches a constant at high enough SNR. This means that the gap between the coded and uncoded error curves remains approximately constant at reasonably high SNR. Intuitively, coding gain can be interpreted as the amount of additional transmit power required to achieve a particular P_{e} in an uncoded system (versus a coded system).

3.2 Measuring Coding Gain

In this lab experiment, you will use the intuitive interpretation of coding gain to empirically estimate the coding gain of a rate $\left(\frac{1}{3}\right)$ repetition code. The questions below will guide you through the process of approximating the coding gain of your repetition code. Set the following parameters in your system. Leave all unspecified parameters set to their default value.

- TX sample rate = 20 MSamp/sec
- TX oversample factor = 8
- RX sample rate = 10 MSamp/sec
- RX oversample factor = 4
- Packet length = 1000 bits
- Capture time = 1 msec

There are several places where bit errors can occur in your wireless communication system. In the Lab 7 you learned how critical frame detection errors can lead to an error rate of approximately $\frac{1}{2}$. You have also learned about frequency synchronization errors in OFDM systems, and how large frequency offsets can also lead to severe error rates. In this experiment, you will only be interested in errors caused by additive noise. That is, you should disregard any trials in which an error might have occurred from a critical error in frame detection or a severe frequency offset (i.e., throw out trials in which severe error rates are observed).

Questions

Be sure to answer the following questions in the lab itself, as this lab does not require you to submit a lab report.

1. Estimate the minimum prewireless channel SNR required by the coded system to maintain an error rate below 10^{-2} (i.e., maintain an error rate on the order of 10^{-3}). You will be doing this by changing the artificial noise power added to your signal before being transmitted. This will be done from the front panel of *top_tx.vi* inside the *channel model parameters* control cluster. You will need to transmit multiple packets to verify that your estimated SNR reliably produces the desired error rate.

2. Estimate the additional pre-wireless channel SNR required by the uncoded system to maintain a similar error rate. To do this, disable the channel coding feature on the front panels of *top_tx.vi* and *top_rx.vi*. Also set the *Packet length* parameter to 3000. You will need to transmit multiple packets to verify that your estimated transmit power reliably produces the desired error rate.

3. Based on the SNRs you have estimated in the previous two problems, what is the coding gain (in dB) of the resulting system?

   Hint: X dBm − Y dBm = (X − Y) dB.

3.3 Lab Turn In

Demonstrate to the instructor that your code is working properly. Show the instructor that the uncoded system requires higher SNR to maintain a comparable error rate (i.e., versus the coded system).

DIRECTIONS FOR LAB SUBMISSION

This lab does not require any lab report. Please be sure to answer all the lab questions and show your results to the instructor.

Answer the following questions about your channel coding experiment.

1. Does the coding gain you measured in lab correspond to the coding gain you observed through simulation in the pre-lab?

2. Using channel coding adds resilience to a digital communication system. Discuss the tradeoffs associated with using coding, specifically that of rate and performance. If you could adaptively decide when to use coding (i.e., depending on the received SNR), would you use coding when the SNR is high, low, or always and why?

Appendix A

Reference for Common LabVIEW VIs

Appendix A enumerates some of the common VIs you will have to use in this lab and how to access them (i.e., the path to get to them from the **All Functions** tool palette). Note you can also use LabVIEW QuickDrop to search for VIs by name by clicking `<CTRL> + <SPACE>` when you are on a LabVIEW diagram window. For additional information about each VI, place the VI in a LabVIEW block diagram, right-click on the VI, and select *Help*. This will bring you to a detailed description of the VIs behavior as well as its inputs and outputs.

Insert Into Array.vi	**Programming⇒Array**
Interleave 1D Arrays.vi	**Programming⇒Array**
Max & Min.vi	**Programming⇒Array**
Unbundle By Name.vi	**Programming⇒Cluster & Variant**
Tick Count (ms).vi	**Programming⇒Timing**
Random Number (0-1).vi	**Mathematics⇒Numeric**
Complex To Polar.vi	**Programming⇒Numeric⇒Complex**
Complex To Re/Im.vi	**Programming⇒Numeric⇒Complex**
Case Structure.vi	**Programming⇒Structures**
Local Variable.vi	**Programming⇒Structures**
Convolution.vi	**Signal Processing⇒Signal Operation**
FFT.vi	**Signal Processing⇒Transforms**
Inverse FFT.vi	**Signal Processing⇒Transforms**
Mean.vi	**Mathematics⇒Prob & Stat**
A x B.vi	**Mathematics⇒Linear Algebra**
Inverse Matrix.vi	**Mathematics⇒Linear Algebra**
Generate Bits.vi	**Modulation⇒Digital**
Fractional Resample.vi	**Modulation⇒Analog⇒Utilities**
Generate Filter Coefficients.vi	**Modulation⇒Digital⇒Utilities**
Align To Ideal Symbols.vi	**Modulation⇒Digital⇒Utilities**

Bibliography

[1] Breed, G., "Analyzing Signals Using the Eye Diagram," *High Frequency Electronics*, pp. 50–53, November 2005.

[2] Heath Jr., R. W., *EE 371C / EE 381V: Wireless Communications Lab*, http://www.profheath.org/teaching/ee-371c-ee-381v-wireless-communications-lab/, accessed September 2011.

[3] Heath Jr., R. W., *Introduction to Wireless Digital Communication: A Signal Processing Perspective*, course notes for EE 371C taught at The University of Texas at Austin.

[4] IEEE-SA Standards Board, IEEE Std 802.11a-1999 (R2003)—*Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specification. High Speed Physical Layer in the 5 GHz Band*, June 12, 2003, from http://standards.ieee.org/

[5] National Instruments, *LabVIEW Fundamentals*, August 2005, retrieved December 14, 2005 from http://www.ni.com/pdf/manuals/374029a.pdf.

[6] Johnson, Richard C. and Sethares, William A., *Telecommunications Breakdown: Concepts of Communication Transmitted via Software-Defined Radio*, Prentice Hall, 2003

[7] Moose, P., "A technique for orthogonal frequency division multiplexing frequency offset correction," *IEEE Trans. Commun.*, vol. 42, no. 10, pp. 2908–2914, October 1994.

[8] National Instruments, *Bandwidth, Sample Rate, and Nyquist Theorem*, retrieved September 13, 2011, from http://zone.ni.com/devzone/cda/tut/p/id/2709.

[9] National Instruments, *Getting Started with NI LabVIEW Student Training*, June 2010, retrieved September 13, 2011, from http://zone.ni.com/devzone/cda/tut/p/id/7466.

[10] National Instruments, *How Can I Learn LabVIEW?*, retrieved September 13, 2011, from http://www.ni.com/gettingstarted/labviewbasics.

[11] National Instruments *LabVIEW 101*, retrieved September 13, 2011, from http://www.ni.com/lv101.

[12] National Instruments, *LabVIEW Fundamentals*, August 2007, retrieved September 23, 2011, from http://www.ni.com/pdf/manuals/374029c.pdf.

[13] National Instruments *NI USRP information*, retrieved September 13, 2011 from http://www.ni.com/usrp.

[14] National Instruments, *Understanding RF & Microwave Specifications—Part I*, retrieved September 13, 2011, from http://zone.ni.com/devzone/cda/tut/p/id/4446.

[15] National Instruments, *Understanding RF & Microwave Specifications—Part II*, retrieved September 13, 2011, from http://zone.ni.com/devzone/cda/tut/p/id/4448.

[16] Schmidl, T. M. and Cox, D., "Robust frequency and timing synchronization for OFDM," *IEEE Trans. Commun.*, vol. 45, no. 12, pp. 1613–1621, 1997.

[17] Wikipedia *Software-defined radio*, retrieved September 13, 2011, from http://en.wikipedia.org/wiki/Software_radio.

nts
NATIONAL TECHNOLOGY & SCIENCE PRESS