U.S.N.A. — Trident Scholar project report; no. 300 (2002)

DEVELOPMENT OF A DIGITAL SIGNAL PROCESSOR (DSP) BASED CHAOTIC COMMUNICATION SYSTEM WITH EMPHASIS ON MILITARY APPLICATIONS

by

Midshipman Noah F. Reddell, Class of 2002
United States Naval Academy
Annapolis, Maryland

Certification of Advisers Approval

Associate Professor Erik M. Bollt
Mathematics Department

CDR Thaddeus B. Welch, III, USN
Electrical Engineering Department

Acceptance for the Trident Scholar Committee

Professor Joyce E. Shade
Deputy Director of Research & Scholarship

USNA-1531-2

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. REPORT DATE (DD-MM-YYYY)	2. REPORT TYPE	3. DATES COVERED (FROM - TO)
07-05-2002		xx-xx-2002 to xx-xx-2002

4. TITLE AND SUBTITLE
Development of a digital signal processor (DSP) based chaotic Communication System with Emphasis on Military Applications
Unclassified

5a. CONTRACT NUMBER
5b. GRANT NUMBER
5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Reddell, Noah F. ;

5d. PROJECT NUMBER
5e. TASK NUMBER
5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME AND ADDRESS
U.S. Naval Academy
Annapolis, MD21402

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME AND ADDRESS
,

10. SPONSOR/MONITOR'S ACRONYM(S)
11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
APUBLIC RELEASE
,

13. SUPPLEMENTARY NOTES

14. ABSTRACT
See report.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE	Public Release	111	email from USNA, Annapolis, MD, (blank) lfenster@dtic.mil
Unclassified	Unclassified	Unclassified			19b. TELEPHONE NUMBER International Area Code Area Code Telephone Number 703767-9007 DSN 427-9007

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATE COVERED
	7 May 2002	

4. TITLE AND SUBTITLE
Development of a digital signal processor (DSP) based chaotic communication system with emphasis on military applications

5. FUNDING NUMBERS

6. AUTHOR(S)
Reddell, Noah F. (Noah Freeman), 1981-

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)		8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
US Naval Academy Annapolis, MD 21402		**Trident Scholar project report no. 300 (2002)**

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT	12b. DISTRIBUTION CODE
This document has been approved for public release; its distribution is UNLIMITED.	

13. ABSTRACT: Chaotic systems are aperiodic, deterministic, and sensitive to slight variations in initial condition. The latter property presents the problem that behavior of the system cannot be predicted for a significant period of time, even though for the next instant, it is completely predictable. The frequency domain and time domain properties of chaotic systems have generated interest in the military communications field. This paper explores the advantages of a communication system based on a chaotic carrier for military purposes. Digital signal processing techniques are used to implement the system, perform frequency domain analysis, and generate system performance curves. A series of computer simulations were used to enhance system performance. A digital transmitter was developed that is difficult to localize and detect by exploiting some of the natural properties of chaotic systems. A new dual synchronizing receiver scheme is demonstrated that works by storing samples over an entire bit period prior to estimation. Results show significantly better bit error probability performance in comparison to previously published methods. Research concentrated on developing a system that works effectively without a comspicuous signature in the frequency domain. Such a system should be extremely useful for cover wireless communications.

14. SUBJECT TERMS: **Chaos communication, digital signal processing, Lyapunov exponent, dual synchronizing response system**	15. NUMBER OF PAGES **111**
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT

Abstract

Chaotic systems are aperiodic, deterministic, and sensitive to slight variations in initial condition. The latter property presents the problem that behavior of the system cannot be predicted for a significant period of time even though for the next instant, it is completely predictable. The frequency domain and time domain properties of chaotic systems have generated interest in the military communications field. Midshipman Reddell has explored the advantages of a communication system based on a chaotic carrier for military purposes. He used digital signal processing techniques to implement the system, perform frequency domain analysis, and generate system performance curves. A series of computer simulations were used to enhance system performance. He developed a digital transmitter that is difficult to localize and detect by exploiting some of the natural properties of chaotic systems. The use of discrete methods allowed for considerable improvement over earlier schemes. He demonstrated a new dual synchronizing receiver scheme that works by storing samples over an entire bit period prior to estimation. Results show significantly better bit error probability performance in comparison to previously published methods. He also developed a systematic method to further improve the bit error performance and covertness of the system by optimizing the selection of the modulation parameters. This involved a comparison of the average energy per bit and testing each parameter set for stability using the Lyapunov exponent method. Research concentrated on developing a system that works effectively without a conspicuous signature in the frequency domain. Such a system should be extremely useful for covert wireless communications.

Keywords: chaos communication, digital signal processing, Lyapunov exponent, dual synchronizing response system

Acknowledgements

It is with sincere and heartfelt gratitude that I would like to acknowledge the support of my project advisors over the last seventeen months. Yes, it was December 2000 when I first started seriously communicating with them and attempting to develop a project proposal. They started off working with me the way they continue to now. I was given the freedom to identify a project that I wanted to work on and to come up with my own approach to the problem. I know we could have worked more efficiently on several occasions, but I appreciate the experience of doing so much on my own.

Associate Professor Bollt showed me that science and mathematics are fun. His chaos toy collection is impressive. His life lessons and unwavering sense of humor will always be remembered and missed.

CDR Thad Welch, III, USN taught me so much about both the research process and officership. I now know why they call it re-search. His support in acquiring all the project hardware, research space, and financial support allowed me to focus on the research. I know I achieved much more because of that.

I would like to thank the Trident Scholar Committee and especially Professor Joyce Shade for their efforts that allow this very unique research program to continue. Every year they offer support based on years of research experience; I received their suggestions very humbly.

So many others assisted me throughout the year, but I would like to specifically acknowledge a few. All of the staff at the Multimedia Support Center went well beyond their duties helping to make my research presentable. The Electrical Engineering Department secretary, Ms. Bonnie Jarrell, helped out in a pinch many times this year. Daphi Jobe offered a tremendous amount of help acquiring hardware, fixing software problems, and being on call when equipment just would not work. Thank you all.

Contents

Chapter 1

Introduction

Chaos has always existed, but it took a new perspective to reveal it. In processes from scientific experiments to every day life, the assumption is often made that if a system is started under similar initial conditions, it will produce similar results. In fact, we make this assumption often enough for it to be taken for granted. For example, the temperature of a chemical reactant can never be measured exactly. A chemist accepts this limitation, assuming qualities like reaction time will vary only slightly due to measurement error. In the 1960's, Edward Lorenz identified a case where this assumption did not work and became one of the first visionaries in the new field of Chaos to explore why.

In chaotic systems, small variations in initial condition can lead to substantially different outcomes. For this reason, whenever measurement error exists the state of even well understood chaotic systems cannot be predicted significantly into the future. While studying weather models, Edward Lorenz found perhaps the most well-known and commonly experienced example of this idea. Weather forecasters have great difficulty making accurate forecasts more than a few days into the future. This is true no matter how many observation stations are available or how often measurements are recorded. The weather system is too complex and too sensitive to small changes [1]. Lorenz described it as the Butterfly Effect. This idea illustrates that a butterfly flapping its wings here at the Naval Academy may eventually and even relatively quickly affect the weather across the globe.

1.1 Why Chaos?

The idea that anything can be calculated in closed form with enough information and computing power does not apply to chaotic systems. Therefore, a paradigm shift from the conventional approach of science is required. More computing power and more sophisticated measurement techniques only improve near-term predictability. The long-term state of chaotic systems still cannot be calculated.

This is not always a problem. Now that mathematicians and other scientists are looking for and studying chaos, a number of interesting properties of these systems are being discovered. As counterintuitive as it first seems, there are a number of reasons why chaotic systems may actually be useful in the field of communication.

Interest in the field of chaos is growing in academic communities and military research groups. In fact, the Army Research Office (ARO) has gone as far as to recently name communication with dynamical systems one of its seven areas of concentration [2]. Also, the Office of Naval Research (ONR) runs a continuing program investigating the use of nonlinear dynamics, including communication on chaotic carriers.

Chaotic systems have a number of special properties that are worth investigating for communication.

- **Apparent Randomness in the Time-Domain -** Chaotic systems look random in time because they are both aperiodic and bounded. This property is worth exploring for covertness.

- **Natural Synchronization -** Two chaotic system can be made to synchronize with each other relatively easily. Synchronization turns out to be fortuitous for creating a communication system and is a fundamental requirement for our system to work.

- **Interesting Control Applications -** Small influences applied intelligently can dramatically influence a chaotic system. Making small adjustments at opportune moments may allow for the ability to control large amounts of energy. This could be very useful for a high-power transmitter. While this idea is not directly explored in this project, some techniques we develop can be adapted.

One question that frequently comes up is how does this system compare to spread spectrum techniques? After all, there are some apparent similarities. Both systems use pseudo-random techniques and transmit with energy distributed over a wide frequency range. The primary difference is the method by which the carrier signal is generated. We explore the relationship between these two techniques further.

It's always good to try new approaches to the same problem. Spread spectrum systems have some disadvantages. Perhaps a chaotic system will address them. Or, as we suspect, chaos might be best for specific applications. Mass-market personal communications is probably not one of them.

1.2 Project Goals

Our initial focus in developing our chaotic communication system was to improve the transmitter's covertness. In every military community there is some unanswered need to prevent an opponent from locating one's transmitter. With standard transmission systems, electronic warfare techniques that have been in use for many years can easily pick out a radio transmission from the surrounding noise and deduce its position.

These systems are also left more vulnerable to jamming, since they concentrate their energy around one carrier frequency. Essentially, all it takes to jam a transmission is to overpower the real transmitter. The term commonly used for this says it all; the energy required to jam is called burnthrough.

We attempt to hide a transmission by blending it with the background. This equates to hiding the transmission energy below or close to the existing noise floor. Analysis and visualization of this process will make the idea more clear in chapter 9.

We design and investigate our system with the idea of camouflage in mind. In the end, more time is spent and more of our interesting results relate to the challenges of discretely generating chaos, improving system performance, and characterizing the effects of different parameters.

Chapter 2

Chaos

James Yorke offered an intriguing name that took hold for the discipline previously referred to as non-linear dynamical systems. The use of the word *chaos* originated from his 1975 paper "Period Three Implies Chaos."[3] There remains a few different rigorous definitions of the term; however, Steven Strogatz offers a widely accepted colloquial definition. He explains, "chaos is aperiodic long-term behavior in a deterministic system that exhibits sensitive dependence on initial conditions."[4] Chaotic systems develop according to deterministic relationships but appear to behave randomly.

The equations that describe such systems can look deceivingly simple. There are low-order chaotic systems as well as high-order ones. Surprisingly, even low-order chaotic systems exhibit rich variety and beautiful complexity. Simple equations can have complicated behaviors. The system depicted in Figure 2.1, for example, is known as the Lorenz Attractor. This chaotic system is so famous in the field that an image of the 'butterfly wings' can be found in numerous publications ranging from specialized papers to popular magazines. The Lorenz system is heavily used throughout this project.

It is worthwhile to explain each of the properties that make up chaos.

2.1 Deterministic Systems

One fundamental concept for appreciating chaotic systems is that they are not based on random processes. When choosing between two paths at a fork, a random or stochastic process would choose one based only on some probability. No definite prediction can be made. On the other hand, to fit the definition above, chaotic systems must be *deterministic*. If they exist in a discrete space, a map can relate one state to the next. If they are continuous, a function can describe the flow. Therefore, knowing enough about a system's state and how it develops, we could exactly predict what path it will take. More frequently, we encounter chaos without knowing a function or map that describes how it develops.

Using the weather as an example again, the system is too big and has too many variables to be written down in the form of an equation. Lorenz's system is merely a model for one of many processes. The Lorentz equations model the convection and rolling of a fluid being heated at the bottom of a container and cooled at the top.

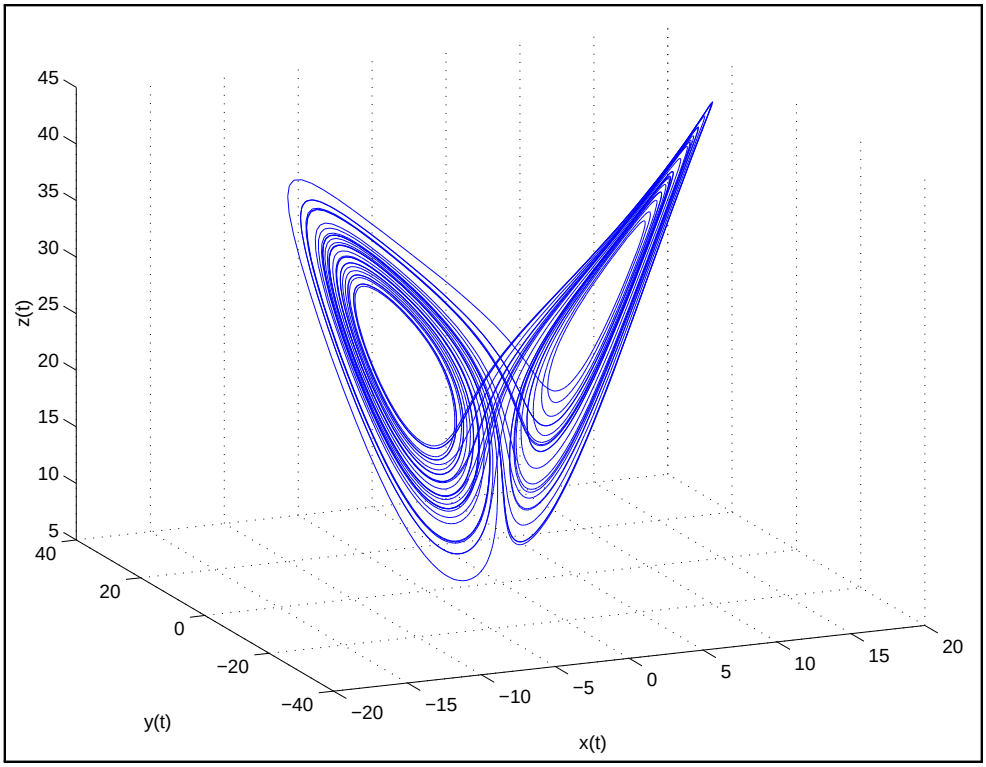

Figure 2.1: The phase portrait associated with the Lorenz Attractor with parameters $\sigma = 10$, $\rho = 28$, and $\beta = 8/3$.

Definition 2.1.1. f $\mathbb{R}$ $\{x_1, x_2, \dots\}$
f **chaotic**
1 $\{x_1, x_2, \dots\}$
h x_1

Definition 2.1.2. $\mathbf{F_t}\ \mathbf{v_0}$ $\dot{\mathbf{v}}$ $\mathbf{f}\ \mathbf{v}$ $\mathbf{v_0} \in \mathbb{R}^n$
$\mathbf{F_t}\ \mathbf{v_0}$ **chaotic**
1 $\mathbf{F_t}\ \mathbf{v_0}$, $t \geq 0$
$\mathbf{F_t}\ \mathbf{v_0}$
$\omega\ \mathbf{v_0}$

2.2 Sensitive Dependence on Initial Conditions

The second and equally important property of chaotic systems is sensitive dependence on initial conditions. The consequence of this property is that even if a perfect model for the global weather system that took into account every possible variable was known, it would still not be useful for long term weather prediction. This is because small errors in measurement can never be completely eliminated. A weather station might report the temperature to be $25.00°C.$ instead of $25.00001°C.$ In chaotic systems, these small errors greatly affect the future behavior of a system. Figure 2.2 illustrates this concept.

Sensitive dependence on initial conditions means solutions of a system are dramatically effected by small changes in their initial condition. Bounded systems will exhibit complex aperiodic behavior with unstable orbits. Such systems are referred to as chaotic. Later, we need a metric to judge whether or not a particular system is chaotic. As a matter of terminology, a chaotic system has a chaotic orbit or flow.

A statistic called the Lyapunov exponent measures the exponential rate at which the error between two nearby solutions of a system grows or decreases. If this error grows on average, or the Lyapunov exponent is positive, then sensitive dependence on initial condition is exhibited. If the error decreases and both initial conditions settle into the same solution, then the Lyapunov exponent is negative and the solution is stable. Definition 2.2.1 below, formally defines sensitive dependence on initial conditions for a map, or discrete system [5]. In chapter 5 we provide more details about the Lyapunov exponent for a flow, and how it can be calculated for a known system.

Definition 2.2.1. Let f be a map on $\mathbb{R}$. A point x_0 has **sensitive dependence on initial conditions** if there is a nonzero distance d such that some points arbitrarily near x_0 are eventually mapped at least d units from the corresponding image of x_0. More precisely, there exists $d > 0$ such that any neighborhood N of x_0 contains a point x such that $|f^k(x) - f^k(x_0)| \geq d$ for some nonnegative integer k.

2.3 Dense Orbits

Since a chaotic system is bounded, solutions remain in a constrained phase space. The Lorenz system, for example, exists in a fixed three-dimensional space that looks like a pair of wings. Since the system is both aperiodic and deterministic, every point in that space must eventually be touched as the flow evolves. If the same point was ever revisited by a flow, then that flow would be periodic. A *dense orbit* means that between two nearby parts of an orbit, no matter how close together, there are or will be infinitely many other parts to the same orbit sandwiched in between. Figure 2.3 illustrates what happens as the Lorenz system is allowed to evolve for progressively longer periods of time. The idea of a dense orbit begins to develop.

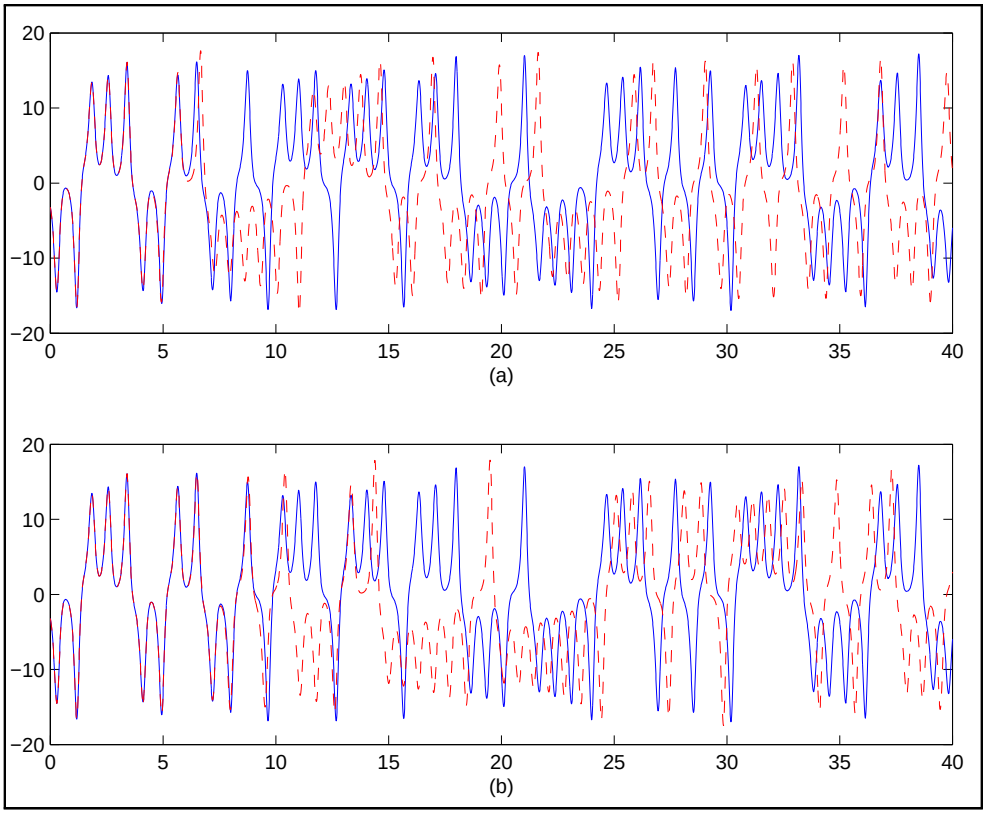

Figure 2.2: Demonstration of sensitive dependence on initial condition. Time series view of $x(t)$ term for two systems started with similar initial conditions. (a) Initial x value -3.20 and -3.21 (b) Initial x value -3.20 and -3.201. Notice that a ten-fold improvement in accuracy of the two starting points does not significantly increase the time for which the two systems follow similar orbits.

2.4 Chaotic Attractors

Chaotic attractors

$\mathbb{R}^3$

Definition 2.4.1. f x_0 **forward limit set** $\{f^n\ x_0\ \}$

$\omega\ x_0$ $\{x$ N ϵ $n > N$ $|f^n\ x_0\ - x| < \epsilon\}$.

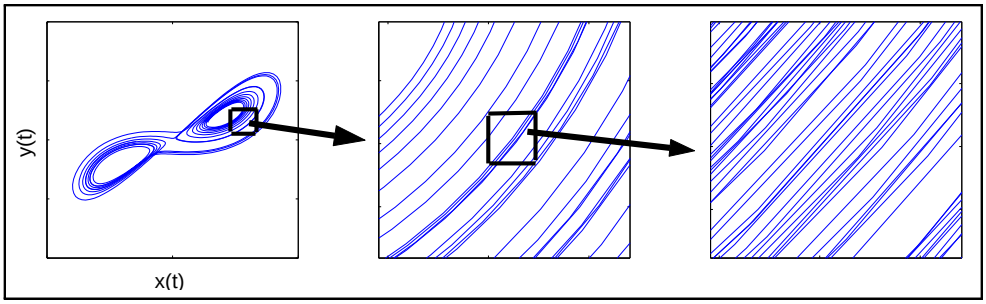

Figure 2.3: Illustration of a dense orbit. The three plots show the same flow of the Lorenz system. Each is started with the same initial condition, but iterated for progressively longer time periods. The square areas indicated are expanded in the plot to the right to show that given more time, the flow will pass an infinite number of times through a small volume. Note that any apparent intersections of orbits actually have different $z(t)$ values because the system is aperiodic.

$\omega\ x_0$ x_1

$\{f^n\ x_0\ \}$ x_1 **attracted to** $\omega\ x_1$ $\omega\ x_1$

$\omega\ x_0$

an orbit or flow is chaotic if it's not asymptotically periodic, is bounded, and the Lyapunov exponent of the system is greater than zero

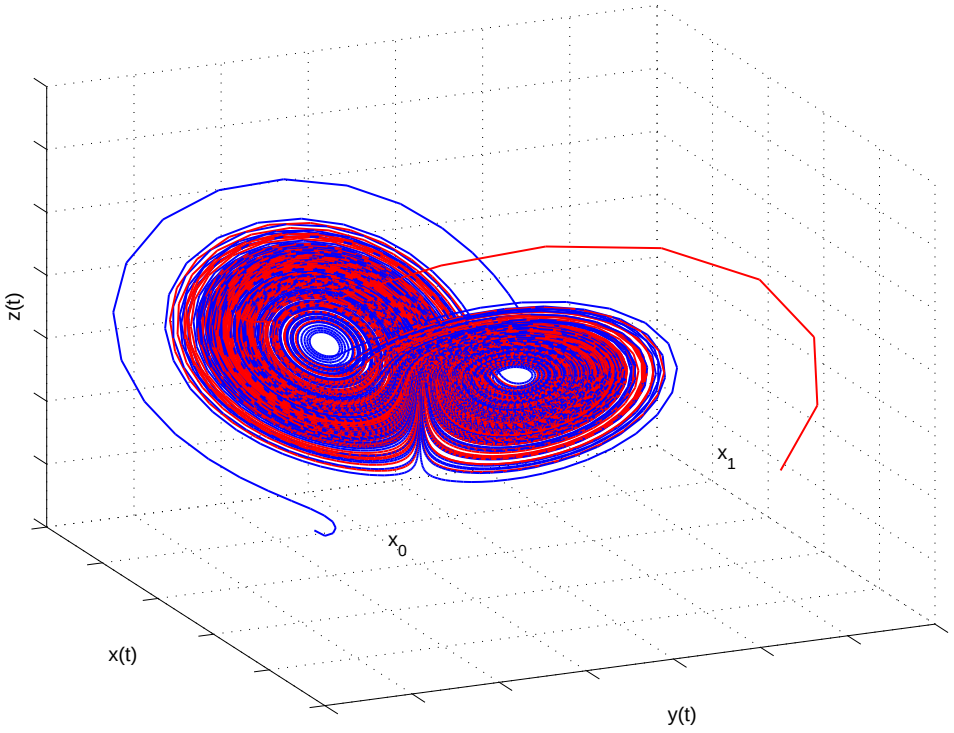

Figure 2.4: Illustration of an attractor. Flows of the two initial conditions x_0 and x_1 both end up in the same forward limit set.

Chapter 3

Communication Scheme

Our system for sending a message using a chaotic signal as a carrier is largely based on work shown by Cuomo, et al. in [7]. Our improvements are motivated by the desire to discretely implement their system and to improve it's performance from a communications perspective. Our goal while doing this is to create a system specialized for covert transmissions or signal camouflage.

Several schemes for communication using chaos have been tried. See [7],[8], [9], [10], [11], and [12].

The one we determined would be most useful for discrete implementation and covert transmission is parameter modulation.

3.1 Parameter Modulation

Parameter modulation works by first having two identical chaotic systems – one running in the transmitter and one in a receiver. By synchronizing the two, their instantaneous states are very close. (Synchronization is discussed in chapter 4).

A binary message is sent by either maintaining the synchronization or breaking it. When the receiver is synchronized with the transmitter over some defined bit period, this is interpreted as a zero-bit. If over the same bit period, the receiver is out of synchronization with the transmitter, this is interpreted as a one-bit.

In a real communication system, the receiver is separated by a significant distance from the transmitter. All it can know for sure is its own state and a single influence received from the transmitter either by wire or wirelessly. Therefore, the determination of whether the systems are synchronized or not has to be made using only this information.

This can be done by using an error term comparing the received influence signal with what the system would expect,

$$e_x(t) = x_{influence} - x_{prediction}. \tag{3.1}$$

The $x_{influence}$ term would be the output of the transmitter altered by noise and other channel effects. The $x_{prediction}$ term is what the receiver is expecting to receive based on its own parameters and state.

This instantaneous error must be evaluated over an entire bit period. One way to do this is to evaluate the integral of the absolute value of the error term.

$$\text{Bit Error} = \int_{t_0}^{t_0+T_{bit}} e^2(t)dt. \tag{3.2}$$

To do this in an electrical circuit and determine the received bit would require some type of low pass filter and a comparison of the bit error with a threshold value. With the unavoidable presence of noise, even two perfectly matched parameter sets would still lead to some observed error. Our implementation on discrete hardware offers some advantages for this comparison process.

3.2 Discrete Implementation

We could implement a discrete system analogous with the one described above. Instead of a continuous comparison of the influence signal with the corresponding receiver state, the comparison would be done at a fixed sample rate.

$$\text{Bit Error} = \sum_{n=0}^{N} e^2[n]. \tag{3.3}$$

However, this would not take advantage of the added flexibility of running the system on a microproccesor.

Our discrete chaotic transmitter operates in the same fashion as the continuous version described previously. At the beginning of every bit period, the transmitter chooses between two parameter sets based on the message bit to be sent. One term of the chaotic system is sent through the medium to the receiver.

The first discrete chaotic receiver we developed also operates very similarly to the continuous one described. The error at each sample is stored in the processor's memory over the duration of a bit period. At the end of each bit, the stored error values are squared and summed. This sum is compared to a predetermined threshold value. Bit Error less than the threshold causes the signal to be interpreted as a zero-bit and greater than the threshold is interpreted as a one-bit. The threshold value can be set half-way in between the error expected due to channel noise and sampling errors when the systems are perfectly synchronized and this error in addition to error experienced when the two systems do not have matching parameter sets.

3.3 Improvements

Similar to most processors, the digital signal processors used to implement the chaotic transmitter and receiver in this project have memory to store samples and can perform complex logical operations. We take advantage of these capabilities.

Our new **dual synchronizing response system** corrects two issues that arise when using the basic scheme described above. First, the above scheme assumes that at the beginning of a bit period the systems are synchronized. In that case, without any noise, a zero-bit would cause the bit error to be zero. In reality, the systems would not start off synchronized due to noise or if the previous bit was a one-bit. To make up for this, the bit period must be lengthened so that the systems can synchronize. This will result in a lower data rate.

The second problem is that the transmitter cannot use radically different parameter sets without further lengthening the bit period. Using the parameter modulation scheme, an ideal set of parameters would immediately cause perfect synchronization when the transmitter and receiver parameters matched, and immediately large error term when they were mismatched.

We notice that choosing different parameter sets does not totally break the synchronization. It causes the systems to never exactly match, but the receiver still tends to follow the transmitter to some extent.

To address both of the above problems, we run two response systems simultaneously on the same receiver processor. One response system parameter set corresponds to a zero-bit and the other corresponds to a one-bit. Both systems attempt to synchronize with the drive system over the entire bit period. Then, the sum of errors, as in eqn. 3.3, experienced by both response systems are compared. The system with less error determines the received bit and both response system states are updated to reflect the better match. This process requires memory and comparison ability not easily available in a simple component based implementation. By taking advantage of the abilities of DSP hardware, we have achieved better performance than the system in [7].

Figure 3.1 shows the four cases of our dual synchronizing receiver system. The drive system chooses parameter set A or B based on the message bit. The plots show the drive system and how the two response systems (one using set A and one using set B) respond. We desire that a matched set of parameters between the transmitter and receiver cause a quick and tight coupling while a mismatched set lead to large error.

Figure 3.2 shows a single bit window used in our dual synchronizing receiver scheme. This particular window is 100 samples long. The influence signal from the transmitter is affected by additive Gaussian noise and the two receiver versions attempt to synchronize to the influence signal. The error squared is plotted in the bottom subplot. The sum of squares of the error for both receiver systems is used to determine the best match with the influence signal. This comparison is used to estimate which bit was received.

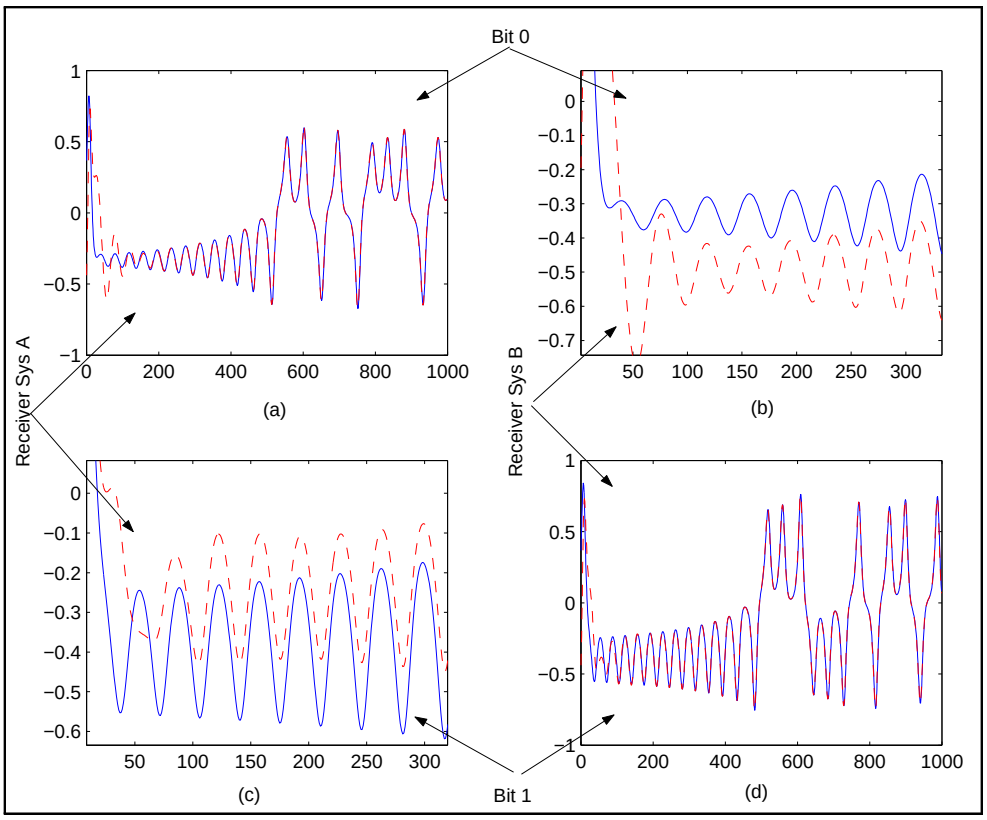

Figure 3.1: All possible combinations of bit sent and receiver system for our dual synchronizing receiver scheme, plotted as voltage vs. sample number. (a), transmitter and receiver use parameter set A (b), transmitter uses set A, receiver uses set B, (c), transmitter uses set B, receiver uses set A (d), transmitter and receiver use parameter set B.

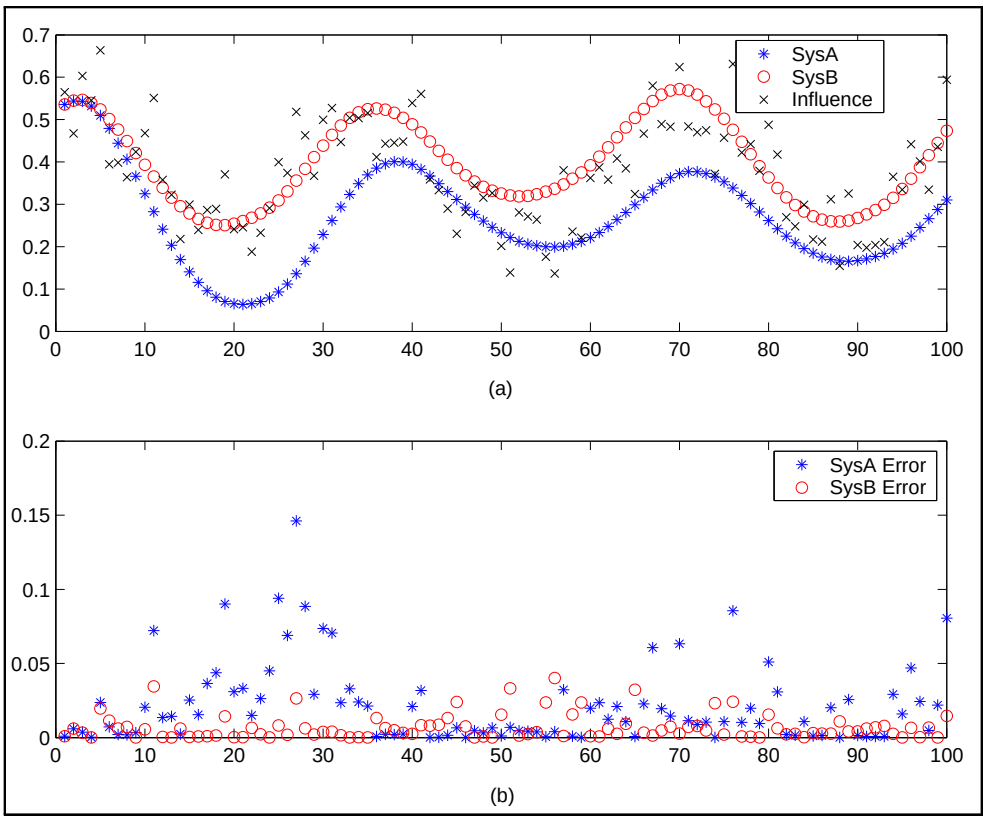

Figure 3.2: Receiver evaluation of a particular bit, (a) Influence signal with noise effects and attempts to synchronize by both receiver systems (Volts vs. Sample Number), (b) Error^2 between both receivers and the influence signal (Volts^2 vs. Sample Number).

Chapter 4

Synchronization

With the properties of chaos already discussed, it seems strange that chaotic systems could be used for communication systems. How can a message be sent if the receiver does not know the state the transmitter? The particular scheme we use to transmit a message has been discussed in greater detail in chapter 3. Now, we analyze a surprising property of some chaotic systems. This is the natural ability to synchronize to an identical system by sharing only one state variable [13],[11].

We consider the famous Lorenz System:

$$\begin{aligned} \dot{x} &= \sigma(y - x), \\ \dot{y} &= \rho x - y - xz, \\ \dot{z} &= xy - \beta z. \end{aligned} \tag{4.1}$$

The parameters σ, ρ, and β have been removed from their original context in Lorenz's convection process but they are still significant for our purposes. It turns out that the Lorenz system given above has a dynamic range (range of x, y, and z solutions) that is impractical for the Digital-to-Analog and Analog-to-Digital converters (CODECs) attached to our DSPs. Additionally, the system evolves at a impractical rate for the bandwidth of our CODECs. For these reasons, we will use a magnitude and time scaling change of variables.

The PCM3006 CODECs in our hardware are capable of operating between -1V and +1V. They take 48,000 discrete samples per second. By the Nyquist Sampling Theorem (Theorem 4.0.1 [14]), they can accurately sample and reconstruct signals with a maximum frequency of 24 kHz.

Theorem 4.0.1. *Let $x(t)$ be a bandlimited signal with $X(f) = 0$ for $|f| > f_M$. Then $x(t)$ is uniquely determined by its samples $x(nT)$, $n = 0, \pm 1, \pm 2, \ldots$, if $f_s > 2f_M$, where $f_s = \frac{1}{T}$. T is the sampling period and f_M is the highest frequency of the signal. [14]*

We will choose a scaling factor by $\frac{1}{A}$ that allows the x term to be sent to the Digital-to-Analog converter without saturation. The system will be sped up or slowed down by a time scale T_S that allows efficient use of the available 24kHz CODEC bandwidth. These terms

will need to be adjusted based on the particular parameters chosen and the time scaling will be tied to the step size of the differential equation solver. The variable $u(t)$ will represent the amplitude and time scaled signal to be transmitted.

$$-1 < u(t) < +1, \text{ and } U(f) = 0 \text{ for } |f| > \frac{f_S}{2}.$$

The uniform scaling is given by the substitution:

$$u = \frac{x}{A}, v = \frac{y}{A}, w = \frac{z}{A}. \tag{4.2}$$

Then,

$$\begin{aligned} \dot{u} &= \frac{\dot{x}}{A} = \frac{\sigma}{A}(Av - Au), \\ \dot{v} &= \frac{\dot{y}}{A} = \frac{1}{A}(A\rho u - Av - A^2uw), \\ \dot{w} &= \frac{\dot{z}}{A} = \frac{1}{A}(A^2uv - A\beta w). \end{aligned}$$

Time scaling by T_S is achieved by the following change of variables:

$$\tau = \frac{t}{T_S} \text{ and, } \frac{d\tau}{dt} = \frac{1}{T_S}\frac{dt}{dt} = \frac{1}{T_S}. \tag{4.3}$$

Utilizing the chain rule, the change of variables in equation 4.3 affects the original system as follows:

$$\begin{aligned} \frac{dx}{d\tau} &= \frac{dx}{dt}\frac{dt}{d\tau} = T_S\frac{dx}{dt}, \\ \frac{dy}{d\tau} &= \frac{dy}{dt}\frac{dt}{d\tau} = T_S\frac{dy}{dt}, \\ \frac{dz}{d\tau} &= \frac{dz}{dt}\frac{dt}{d\tau} = T_S\frac{dz}{dt}, \end{aligned} \tag{4.4}$$

Thus, the time and amplitude scaled drive system (transmitter) is:

$$\begin{aligned} \dot{u} &= T_S\sigma(v - u), \\ \dot{v} &= T_S(\rho u - v - Auw), \\ \dot{w} &= T_S(Auv - \beta w). \end{aligned} \tag{4.5}$$

4.1 Drive - Response Coupling based on Parameter Set Match or Mismatch

In our discrete scheme, we further an idea initiated by Cuomo, et al. [7]. They sent a binary message by flipping the β parameter of the drive system between two values. This adjustment

slightly upsets the synchronization between the drive and response systems. The presence or absence of error at the response system could then be used to determine the message bit.

Coupling is achieved by sharing the u term from the drive system with the response system. Notice in equation 4.6 that u takes the place of u_r in the equations for $\dot{v}_r$ and $\dot{w}_r$. The variable u is the influence signal. This coupling can be realized simply as a wire or with more sophistication, a wireless connection.

We maintain the same influence configuration as used by Cuomo, et al. but add complexity by allowing all three parameters in the response system to be independent of the drive system. The transmitter alters the drive system parameters σ, ρ, and β based on a message bit. Parameters σ_r, ρ_r, and β_r represents the counterpart parameters in the response system. These will either be identical or mismatched. Figure 4.1 shows the coupling scheme used.

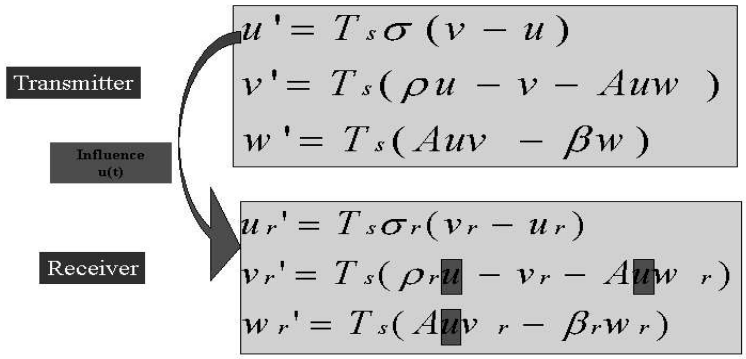

Figure 4.1: Our coupling scheme consists of replacing the u_r term in the receiver with u from the transmitter in the equations for v_r' and w_r'. The parameters may either be matched or mismatched.

The response system receiver is

$$\begin{aligned} \dot{u}_r &\quad {}_S\sigma_r\ v_r - u_r \\ \dot{v}_r &\quad {}_S\ \rho_r u - v_r - \quad uw_r \\ \dot{w}_r &\quad {}_S\quad uv_r - \beta_r w_r \end{aligned} \tag{4.6}$$

rror terms are used to evaluate synchronization

$$\begin{aligned} {}_u &\quad u - u_r \\ {}_v &\quad v - v_r \\ {}_w &\quad w - w_r \end{aligned} \tag{4.}$$

Considering all the error terms at once

$$\mathbf{e} \quad {}_u \quad {}_v \quad {}_w \tag{4.}$$

Taking the derivative with respect to time yields:

$$\begin{aligned}
\dot{e}_u &= (\dot{u} - \dot{u}_r), \\
&= T_S\sigma(v-u) - T_S\sigma_r(v_r - u_r), \\
&= T_S(\sigma v - \sigma_r v_r - (\sigma u - \sigma_r u_r)). \\
\dot{e}_v &= (\dot{v} - \dot{v}_r), \\
&= T_S(\rho u - v - Auw - \rho_r u + v_r + Auw_r), \\
&= T_S(-(v - v_r) - Au(w - w_r) + (\rho - \rho_r)u) \\
&= T_S(-e_v - Aue_w + (\rho - \rho_r)u). \\
\dot{e}_w &= (\dot{w} - \dot{w}_r), \\
&= T_S(Auv - \beta w - Auv_r + \beta_r w_r), \\
&= T_S(Au(v - v_r) - \beta w + \beta_r w_r) \\
&= T_S(Aue_v - \beta w + \beta_r w_r).
\end{aligned} \tag{4.9}$$

4.2 Lyapunov Function Analysis of Stability

Lyapunov functions generalize the idea of potential energy and can be used to characterize the asymptotic behavior of a systems. If we can find a Lyapunov function for the error system above, we can show that it approaches zero over time, and thus the two Lorenz systems synchronize [15]. Again we follow Cuomo's lead and use his Lyapunov function as the basis for the following [7].

$$E(e_u, e_v, e_w) = \frac{1}{2}(\frac{1}{\sigma}e_u^2 + e_v^2 + e_w^2). \tag{4.10}$$

To show synchronization, we want to find that the function $E(e_u, e_v, e_w)$ has a long-term negative slope and so error decreases along solutions of $\mathbf{e}(t)$ in equation 4.8. Taking the derivative with respect to time:

$$\begin{aligned}
\frac{dE}{dt} &= \frac{\partial E}{\partial e_u}\cdot\frac{\partial e_u}{\partial t} + \frac{\partial E}{\partial e_v}\cdot\frac{\partial e_v}{\partial t} + \frac{\partial E}{\partial e_w}\cdot\frac{\partial e_w}{\partial t} \\
&= \frac{e_u\dot{e}_u}{\sigma} + e_v\dot{e}_v + e_w\dot{e}_w \\
&= T_S\{\frac{(u-u_r)}{\sigma}(\sigma v - \sigma_r v_r - (\sigma u - \sigma_r u_r)) \\
&\quad +(v - v_r)(-e_v - Aue_w + (\rho - \rho_r)u) \\
&\quad +(w - w_r)(Aue_v - (\beta w - \beta_r w_r))\}.
\end{aligned}$$

If the drive and response system parameters are the same (**Parameter Set Match**),

$$\begin{aligned}
\sigma &= \sigma_r \\
\rho &= \rho_r \\
\beta &= \beta_r
\end{aligned}$$

then,

$$\begin{aligned}\frac{dE}{dt} &= T_S(e_u e_v - e_u^2 - e_v^2 - \beta e_w^2) \\ &= T_S(-(e_u - \frac{1}{2}e_v)^2 - \frac{3}{4}e_v^2 - \beta e_w^2). \end{aligned} \tag{4.11}$$

Since E is positive definite and $\dot{E}$ is negative definite with $T_S > 0$, Lyapunov's theorem implies $\mathbf{e}(\mathrm{t})$ approaches 0 as $\mathrm{t} \to \infty$. Synchronization will therefore occur. This analysis does not indicate how fast it occurs, but experimentation shows it to be fast enough to achieve a working system.

If the drive and response system parameters not the same (**Parameter Set Mismatch**),

$$\begin{aligned}\sigma &\neq \sigma_r \\ \rho &\neq \rho_r \\ \beta &\neq \beta_r\end{aligned}$$

then,

$$\begin{aligned}\frac{dE}{dt} &= T_S\{\frac{(u-u_r)}{\sigma}(\sigma v - \sigma_r v_r - (\sigma u - \sigma_r u_r)) \\ &\quad +(v-v_r)(-e_v - Aue_w + (\rho - \rho_r)u) \\ &\quad +(w-w_r)(Aue_v - (\beta w - \beta_r w_r))\}.\end{aligned}$$

The derivative above is inconclusive for analysis of the system's ability to synchronize. The next few chapters show our process to optimize the selection of parameter sets for the drive and response systems. This becomes a six dimensional problem and many technical difficulties arise. For certain parameter values the systems do not produce chaotic behavior. In fact, solutions of the Lorenz system, for example, can head towards infinity, reach a steady state, or settle into a periodic orbit if the parameter set is chosen improperly. None of these scenarios are acceptable because they will not work for a covert communication system.

Chapter 5

The Lyapunov Exponent Search

The Lyapunov number measures the average rate of change for error between two nearby solutions of a system. A Lyapunov number greater than one indicates that at any particular time, two orbits near each other will tend to grow apart. If it is less than one, then nearby orbits grow closer and approach equal solutions as time progresses toward infinity. Figure 5.1 illustrates these two cases. The Lyapunov exponent is simply the natural logarithm of the Lyapunov number. Accordingly, if the Lyapunov exponent is positive, then small errors grow and if it is negative, they diminish. It can be used in several analysis related to stability.

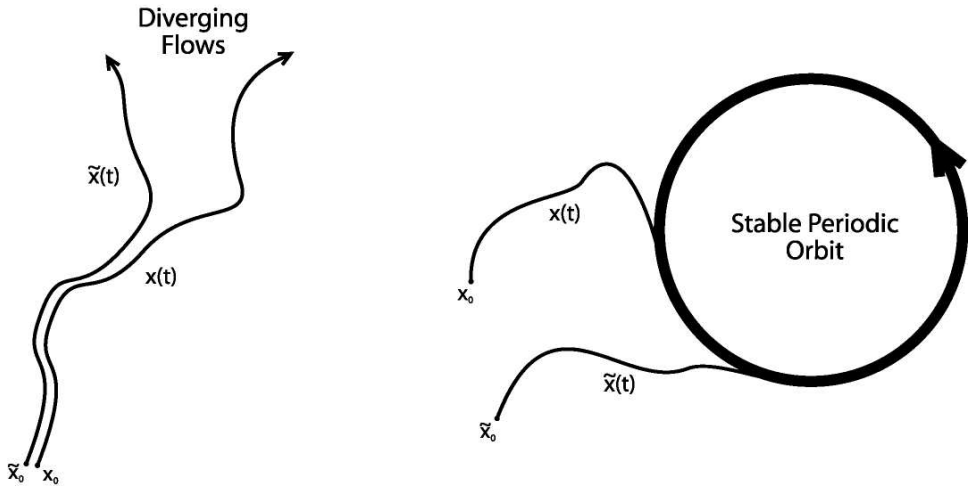

Figure 5.1: Diverging flows and a stable, periodic orbit.

e use the Lyapunov exponent to help identify if a system demonstrates chaotic behav ior for a particular parameter set. ne requirement for chaos is the property of sensitive dependence on initial conditions. An orbit of such a system is unstable so nearby orbits get driven away. If the system is bound to a constrained space and the error grows between two orbits started close to each other, then the systems will be aperiodic and exhibit sensitive dependence on initial conditions. The Lyapunov exponent will be positive.

The Lyapunov exponent is particular to a specific system and set of parameters and it usually can only be approximated by an averaging operation over a long period. Approx

imation of the Lyapunov exponent is easiest for maps since it is referenced as growth rate *per* step of the map.

Definition 5.0.1. Let f be a map of the real line $\mathbb{R}$. The **Lyapunov number** $L(x_1)$ of the orbit $\{x_1, x_2, x_3, \dots\}$ is defined as

$$L(x_1) = \lim_{n\to\infty} (|f'(x_1)| \dots |f'(x_n)|)^{1/n},$$

if this limit exists. The **Lyapunov exponent** $h(x_1)$ is defined as

$$h(x_1) = \lim_{n\to\infty} (1/n)[\ln|f'(x_1)| + \dots + \ln|f'(x_n)|],$$

if this limit exists. Notice that h exists if and only if L exists, and $\ln L = h$. [5]

For flows, evaluation of the Lyapunov number and exponent is more difficult. One method is to transform the continuous system into a discrete one by evaluating the time-1 map of the system. This map represents the state of the system one unit time step later. For a more in depth discussion on computing the Lyapunov exponent for flows, see [16]. We will use the more accessible definition in [5].

Definition 5.0.2. The flow $\mathbf{F}_T(\mathbf{v})$ is the point at which an orbit started at point $\mathbf{v}$ ends up after T time units. The **Lyapunov number** and **exponent** of the flow $\mathbf{v}(t)$ are the Lyapunov number and exponent of the associated time-1 map $\mathbf{F}_T(\mathbf{v})$.

5.1 Estimating the Lyapunov Exponent by the Variational Equation Method

For the flow, $\dot{\mathbf{v}} = f(\mathbf{v}, t)$ we wish to estimate its Lyapunov exponent for a general set of initial conditions. One method for doing this is analogous to the expression given in definition 5.0.1 for maps. We refer to this method as the variational equation method.

Given a particular flow, we will also consider the flow started nearby.

$$\begin{aligned} \dot{\mathbf{v}} &= f(\mathbf{v}, t), \\ \dot{\tilde{\mathbf{v}}} &= f(\tilde{\mathbf{v}}, t). \end{aligned} \tag{5.1}$$

We are interested in quantifying the separation of these two orbit as time progresses so let,

$$\delta_{\mathbf{v}}(t) = \mathbf{v}(t) - \tilde{\mathbf{v}}(t). \tag{5.2}$$

The time derivative of $\delta_{\mathbf{v}}(t)$ is,

$$\begin{aligned} \dot{\delta_{\mathbf{v}}}(t) &= \dot{\mathbf{v}}(t) - \dot{\tilde{\mathbf{v}}}(t), \\ &= f(\mathbf{v},t) - f(\tilde{\mathbf{v}},t), \\ &= f(\mathbf{v},t) - [TSE], \end{aligned} \tag{5.3}$$

where $[TSE]$ represents the Taylor series expansion of $f(\tilde{\mathbf{v}},t)$ about $\mathbf{v}$. The first-order approximation works for our purpose as long as $\tilde{\mathbf{v}}$ is close to $\mathbf{v}$.

$$f(\tilde{\mathbf{v}},t) \approx f(\mathbf{v},t) + f'(\mathbf{v},t)(\tilde{\mathbf{v}}(t) - \mathbf{v}(t)), \text{ for } \tilde{\mathbf{v}} \text{ close to } \mathbf{v}. \tag{5.4}$$

Equation 5.3 above can then be rewritten as:

$$\begin{aligned} \dot{\delta_{\mathbf{v}}}(t) &= f(\mathbf{v},t) - [f(\mathbf{v},t) - f'(\mathbf{v},t)\delta_{\mathbf{v}}(t)] \\ &= Df(\mathbf{v},t) \cdot \delta_{\mathbf{v}}(t). \end{aligned} \tag{5.5}$$

Equation 5.5 is known as the **variational equation**. It describes the growth rate of small errors along an orbit. To estimate the Lyapunov number, we need to solve equation 5.5 for one-unit time steps along with solutions of $f(\mathbf{v},t)$.

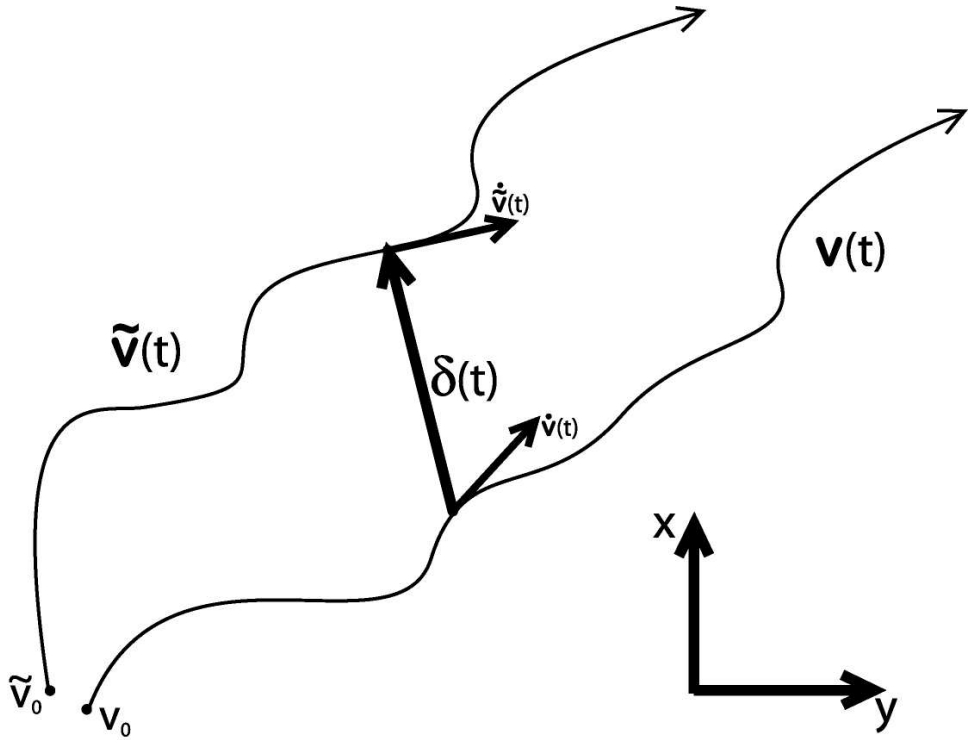

Figure 5.2: Illustration depicting the two flows that contribute to the variational equation, $\mathbf{v}(t)$ and $\tilde{\mathbf{v}}(t)$.

For the Lorenz system,

$$f(\mathbf{v},t) = \begin{cases} \dot{x} &= -\sigma x + \sigma y \\ \dot{y} &= -xz + rx - y \\ \dot{z} &= xy - bz. \end{cases} \tag{5.6}$$

The time derivative of the differential equation (Jacobian matrix) is,

$$Df(\mathbf{v},t) = \begin{bmatrix} -\sigma & \sigma & 0 \\ r - z(t) & -1 & -x(t) \\ y(t) & x(t) & -b. \end{bmatrix}. \tag{5.7}$$

Therefore the variational equation for the Lorenz system is,

$$\dot{\delta_{\mathbf{v}}}(t) = \begin{pmatrix} \dot{\delta}_x \\ \dot{\delta}_y \\ \dot{\delta}_z \end{pmatrix} = \begin{pmatrix} -\sigma & \sigma & 0 \\ r - z(t) & -1 & -x(t) \\ y(t) & x(t) & -b. \end{pmatrix} \cdot \begin{pmatrix} \delta_x \\ \delta_y \\ \delta_z \end{pmatrix}. \tag{5.8}$$

Performing the dot product in equation 5.8 yields the following set of ODEs:

$$\begin{aligned} \dot{\delta}_x &= \sigma(-\delta_x + \delta_y), \\ \dot{\delta}_y &= \delta_x r - \delta_z x(t) - \delta_x z(t) - \delta_y, \\ \dot{\delta}_z &= \delta_y x(t) + \delta_x y(t) - b\delta_z. \end{aligned} \tag{5.9}$$

Note that this set of ODEs must be solved with solutions of $\dot{x}$, $\dot{y}$, and $\dot{z}$. Then our time-1 map will come from simultaneously solving the six ODEs below for one time unit.

$$\begin{aligned} \dot{x} &= -\sigma x + \sigma y, \\ \dot{y} &= -xz + rx - y, \\ \dot{z} &= xy - bz, \\ \dot{\delta}_x &= \sigma(-\delta_x + \delta_y), \\ \dot{\delta}_y &= \delta_x r - \delta_z x(t) - \delta_x z(t) - \delta_y, \\ \dot{\delta}_z &= \delta_y x(t) + \delta_x y(t) - b\delta_z. \end{aligned} \tag{5.10}$$

For our numerical analysis, the error is initially set to:

$$\delta_{\mathbf{v}}(0) = \begin{pmatrix} 1/\sqrt{3} \\ 1/\sqrt{3} \\ 1/\sqrt{3} \end{pmatrix} \text{ or any other vector of unit length and } \mathbf{v}(0) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

5.2 Goals for Lyapunov Exponent Search

Our reason for calculating Lyapunov exponents for the Lorenz system is to identify parameter sets that yield chaos. We need this type of analysis for the bit energy search in chapter 6.

Procedurally, we estimate the Lyapunov exponent of the Lorenz system by evaluating the growth of $\delta_{\mathbf{v}}(t)$ over many time-1 steps. Experimentally, we found that averaging over 1000 time units provides enough accuracy for our purposes. Ultimately, we need to determine if a particular parameter set yields chaos or something else (fixed point, periodicity, unboundedness). Our Lyapunov exponent estimation only needs to be accurate enough to confidently determine if the exponent is positive or negative.

We must establish realistic limits for our search. Ideally, we would like to estimate the Lyapunov exponent for every parameter set in the three-dimensional parameter space. This is simply not possible. As no closed form expression exists, at best only many discrete parameter sets can be evaluated. Each data point requires performing the evaluation noted above for enough time-1 steps that the calculated Lyapunov exponent begins to reliably converge to a value. One data point alone can be computationally intensive. Our chosen calculation length is motivated by modeling the convergence of the calculated Lyapunov exponent as an alternating series. Error will diminish and not exceed earlier errors. We expect that a value settling well after one thousand averages will not suddenly grow away from the average.

5.3 Results of Lyapunov Exponent Search

Throughout this research project, we attempt to calculate the Lyapunov exponent for a total of 1.5 million unique parameter sets. This search amounted to many hours of computation on multiple processors. Still, the largest volume we searched in the parameter space was as follows:

$$\begin{array}{rcccl} -5.0 & < & \sigma & < & 40.0 \\ -5.0 & < & \rho & < & 75.0 \\ -5.0 & < & \beta & < & 20.0 \end{array} \tag{5.11}$$

We noticed quickly that parameter sets with one or more negative parameter were troublesome. Predominately, such sets did not lead to bounded chaos and it was also many times more difficult to estimate the corresponding Lyapunov exponent. The positive range of parameters was limited for the practical reasons already discussed. We chose this particular range because most work that has been done characterizing the Lorenz system involves parameters in this range. For our most thorough exponent estimation, we chose 1,026,127 parameter sets in the region,

$$\begin{array}{rcccl} 0.0 & < & \sigma & < & 35.0 \\ 20.0 & < & \rho & < & 60.0 \\ 0.0 & < & \beta & < & 15.0 \end{array} \tag{5.12}$$

These parameter sets are distributed in a uniform matrix 123 $\times$ 123 $\times$ 63. Figures 5.3 and 5.4 are manipulations of the volume 5.12. Figure 5.5 shows the results for 200,000 parameter sets distributed randomly over a smaller parameter space. The majority (about 2/3) of the parameter sets tested result in negative exponents and are not displayed. The results of the uniform distribution are much easier to use in visualization functions. However, it is important to also consider the randomly distributed points in order to gain some level of confidence that we are not overlooking interesting behavior in between the uniformly spaced samples.

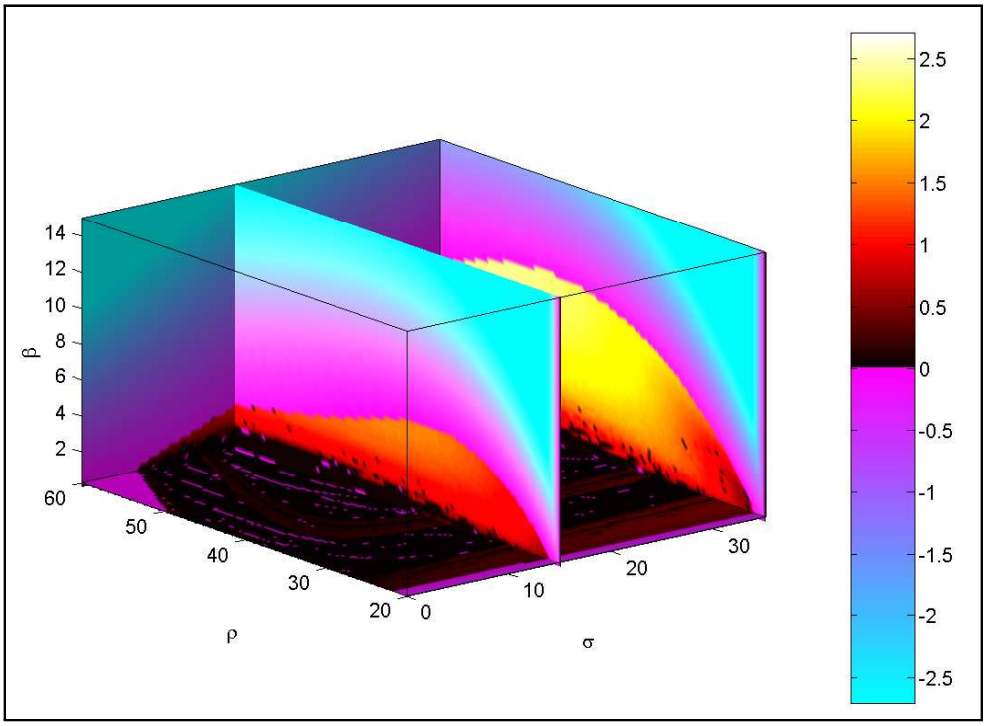

Figure 5.3: Result of Lyapunov exponent estimation for the parameter space shown. It is difficult to visualize values within the volume, so several slices are made through it.

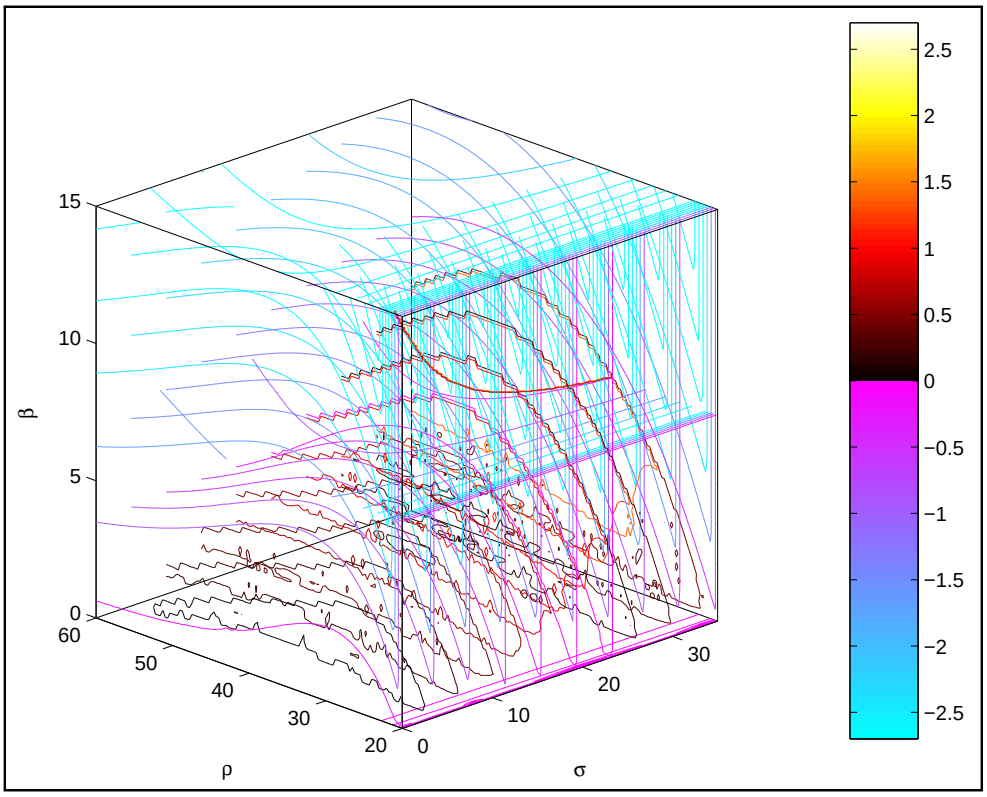

Figure 5.4: Contour plot revealing calculated Lyapunov exponent values inside the uniformly distributed parameter space shown. Variation of exponent value between adjacent parameter sets is not expected to be smooth.

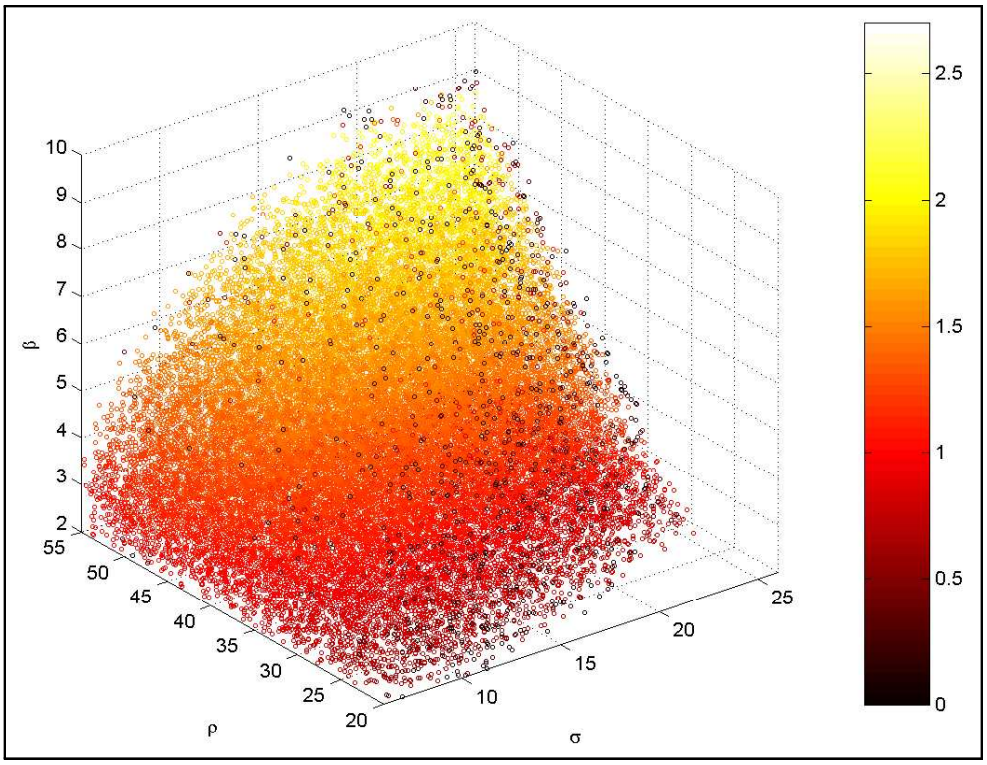

Figure 5.5: Result of Lyapunov exponent estimation for 100,000 randomly distributed parameter sets over the parameter space shown. Only the resulting 33,129 positive exponents are shown.

Chapter 6

Parameter Space Search for Improved Bit Energy

For the Lorenz system, the three parameters σ, ρ, and β alter the behavior of the system [6]. For our dual synchronizing receiver scheme discussed in chapter 3, the transmitter and receiver use two separate sets of the three parameters. One set represents a zero-bit and the other a one-bit. One question that we must explore is how to chose two sets of parameters that yield the best system performance.

Before we can even start our analysis, we must define what we mean by 'best' performance. For our covert communication system, two separate properties must be considered. Our primary objective is to create a transmitter that is difficult to locate. To do this we desire that the transmission frequency spectrum be wide and flat (often referred to as wideband). Distinguishing peaks associated with a periodic transmission (i.e. sinusoids) are easily detectable.

Our secondary goal is to optimize the system's performance in the presence of noise. As with any communications system, when the channel noise that our system transmits through increases, the ratio of successful bits to bits in error degrades.

When we change the parameters to improve one property, another one suffers. For this reason, we must simultaneously compare both of the properties described above when searching for two good parameter sets.

6.1 Improving Bit Error Probability

We find that the error performance of the system is essentially based on the energy of the difference between the one-bit and zero-bit signals. Let,

$$E_{\text{diff}} = \int_{t_1}^{t_2} |u_0(t) - u_1(t)|^2 dt. \tag{6.1}$$

Unlike antipodal communication schemes such as binary phase shift key (BPSK) the energy in a bit is not constant. We cannot solve for this energy in closed form; we can

only observe an average over many bits. To improve the systems error performance when subjected to noise, we wish to maximize this average E_{diff} over all transmitted bits.

Because the DSP hardware gives us the ability to compare two receiver versions against the received signal, we do not have to worry about completely destroying the synchronization by a parameter mismatch which is too great. The parameter matched version will reset the state of both response systems after the bit period. As a result of our system's ability to continue with the best fit system, we are able to maintain synchronization with an aggressive parameter mismatch.

Our initial parameter search was overly simplified. We only allowed the adjustment of the β parameter and only looked at the average bit energy difference E_{diff}. Figure 6.1 shows the results of the rudimentary search. The result was not surprising or tremendously helpful. Mostly, the figure shows that to maximize E_{diff}, $\beta(0)$ and $\beta(1)$ should be as far apart as possible. The problem is that these parameters must be bounded or they cause the system to operate undesirably. (i.e., fixed points, periodic orbits, and unbounded behavior) The initial parameter search did give us an appreciation for the computational complexity of the problem.

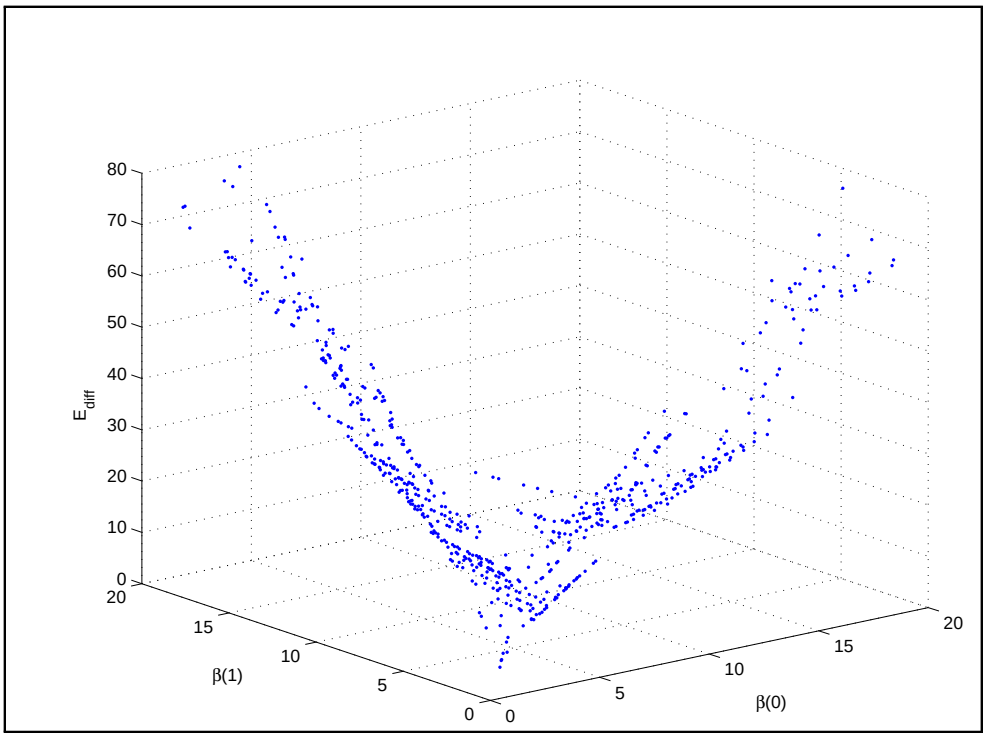

Figure 6.1: Our initial parameter search. The search was restricted to only adjusting the β parameter and only the bit energy difference E_{diff} was taken into account. Results are inconclusive and a more advanced search must be conducted. The dots represent average bit energy difference for the pair of β parameters shown. Generally, the results show that the more different the two β parameters are, the higher is associated bit energy.

Our second method for identifying parameter sets allows for zero-bit and one-bit parameter sets to be mismatched in all three parameters. To greatly simplify the simulation, we fix the zero-bit parameter set to $\sigma = 16.0, \rho = 45.6$, and $\beta = 4.0$. This means that a three dimensional volume of the σ, ρ, and β parameters must be searched for a good one-bit parameter set. Figure 6.2 is one way to view the exterior of the volume in question.

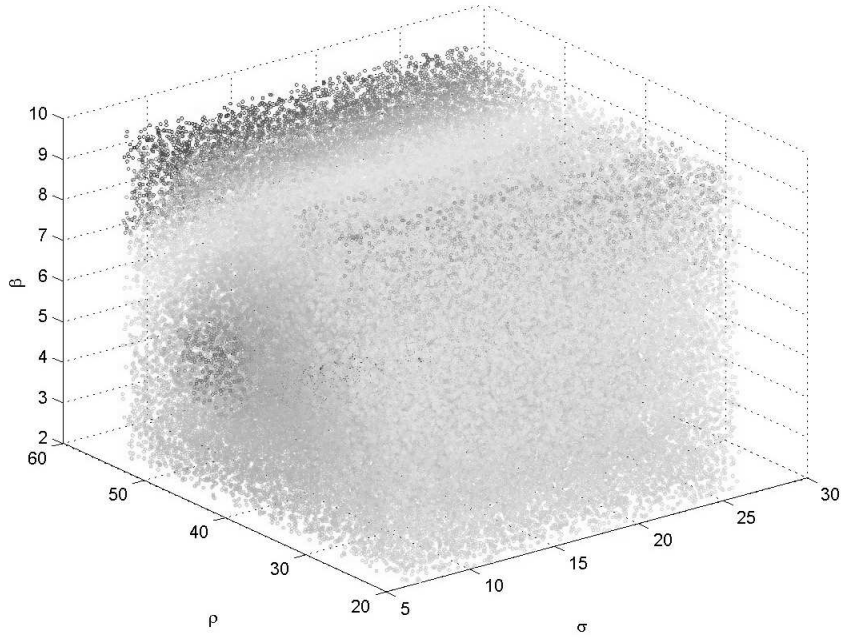

Figure 6.2: Our second bit energy search in three dimensions of the parameter space. One set of parameters is held constant at $\sigma = 16$, $\rho = 45.6$, and $\beta = 4.0$. The other set of parameters is chosen from the 100,000 randomly distributed points shown. Color indicates bit energy so the results imply that parameters nearest the fixed zero-bit set yield the smallest bit energy difference.

6.2 Covert Performance Considerations

At the same time we seek to improve bit energy we wish to maintain or enhance the covertness properties of our system. Due to a large six-dimensional parameter space, our search has to be automated. We have considered two approaches. As one option we could devise an algorithm to look at the shape of the system's average frequency response for each parameter set. This will give us a measure of flatness. Chaotic systems tend to have smooth energy distributions in the frequency domain. Alternatively, we could calculate the Lyapunov exponent for the system using each parameter set.

We balance both qualities to ensure that for a particular parameter set, the system is still chaotic and its energy is smoothly distributed over a wide frequency range. We easily identify cases where a particular parameter set causes the system to reach a steady state (non-chaotic) or some low-order periodic orbit by examining the Lyapunov exponent.

We also explore magnitude and time scaling to ensure that two drive systems operating solely with one or the other parameter sets have similar observable properties in the frequency domain. Amplitude scaling is based on calculating the root mean squared (RMS) value for the drive system under every parameter set used and adjusting to ensure that no matter what parameter set is used, average power output remains the same. Time scaling would affect the chaotic signal's energy distribution in the frequency domain. Ideally, two systems with different parameter sets will look similar in the frequency domain to reduce the chance of bit recovery by a third party. To accomplish this, both systems must distribute signal energy in a similar manner. We did not implement this second type of scaling. Figure 6.3 shows the result of our final bit energy search using amplitude scaling for all of the tested parameter sets. The overall results are quite different from those presented in figure 6.2. The larger bit energy differences presented in figure 6.3 are associated with significant difference in frequency domain properties. This indicates the added need for time scaling in addition to the amplitude scaling we perform.

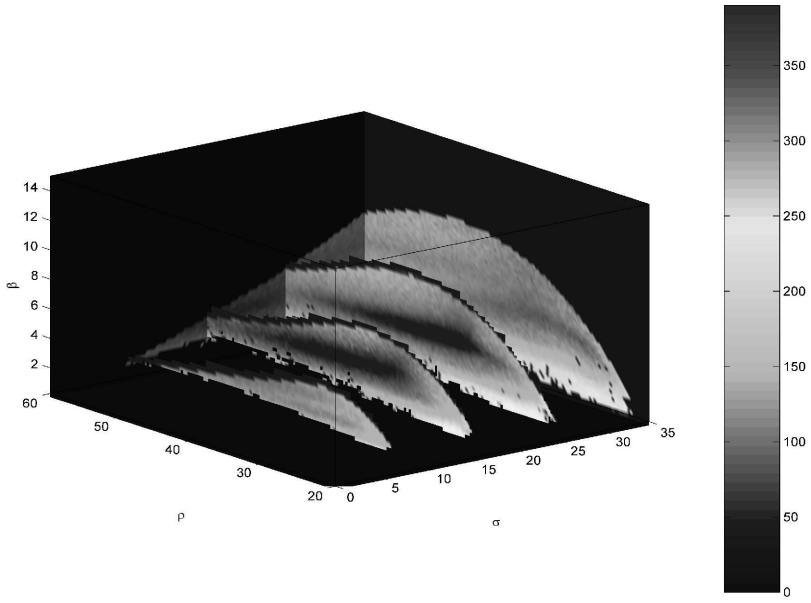

Figure 6.3: Our final bit energy search using the same fixed parameter set as in figure 6.2 and uniformly distributed second parameter sets (all with a pos. Lyapunov exponent). Slice planes are shown to help visualize behavior inside of the cubic volume. Notice the region of low bit energy surrounding the fixed parameter set in the center of the volume.

Chapter 7

Challenges of a Discrete Hardware System

All of the differential equations discussed are continuous systems and can take on any value. We must consider what needs to be modified in order to run these systems on discrete hardware and also if any fundamental changes occur.

Ordinary differential equations (ODEs) like the ones we are using for the Lorenz system can be solved numerically by several algorithms. We select the Runge-Kutta (RK) fourth order algorithm because it yields accurate results relative to its processing requirements [4]. Instead of solving the systems for every instant of time, the Runge-Kutta algorithm only solves the system state at fixed intervals. The chief issue we have faced when transforming the continuous systems to a discrete environment is determining what the step size of that interval should be. (See appendix A for more information on the Runge-Kutta 4-5 Algorithm.)

The transmission system takes one step via the RK algorithm every time the digital-to-analog converter interrupt service routine is called. This rate is fixed at 48kHz by our CODECs. The system must be sped up or slowed down by adjusting the RK step size or by adjusting the time scale T_S, which are a related pair. To most effectively utilize the available bandwidth of the CODEC without aliasing, we have found that the limiting factor is this step size - T_S pair. (The two are closely related.) This is because taking a step that is too big causes the RK algorithm to fail and the discrete system does not emulate its continuous model. In the end, we have a system that is sampled at a rate greater than what Nyquist would require but is necessary for the numerical solver to yield accurate results. We do not experiment with discarding unnecessary samples due to the complications it would add at the receiver.

7.1 Texas Instruments 'C6711 Digital Signal Processor

For this project we use two high-speed digital signal processors (DSPs) to numerically solve the Lorenz System and coordinate input from and output to the CODEC. These processors are highly specialized for performing manipulations on signals and operating on arrays of

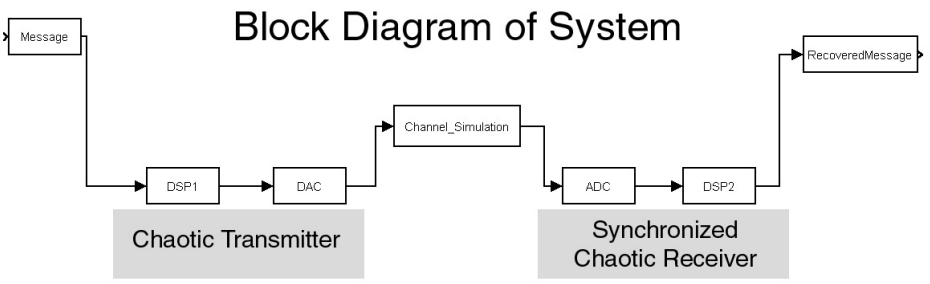

Figure 7.1: Block diagram of DSP based chaotic communication system.

3

9

3

3

3

± 9

93

7.2 Test Equipment

3

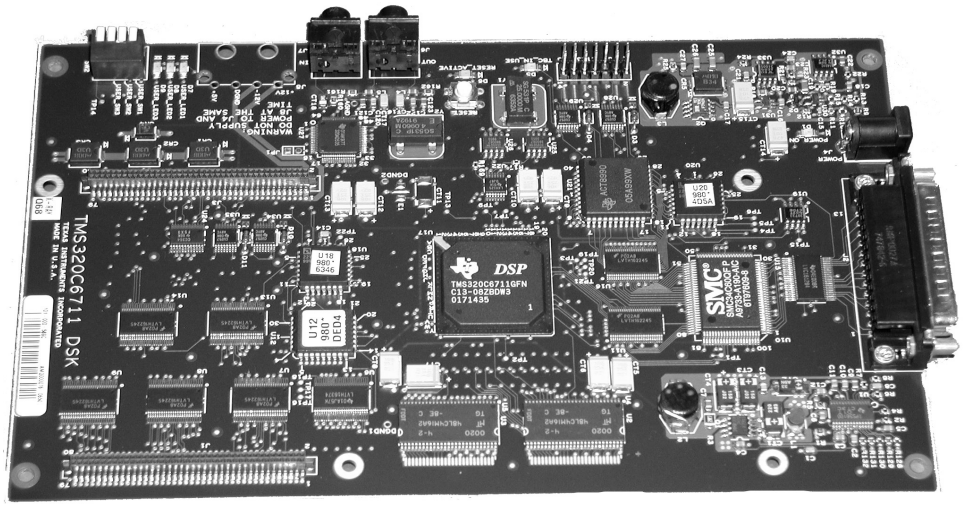

Figure 7.2: Image of Texas Instrument's 'C6711 Digital Signal Processor mounted on a development board. One of these boards implements our transmitter and another implements our receiver.

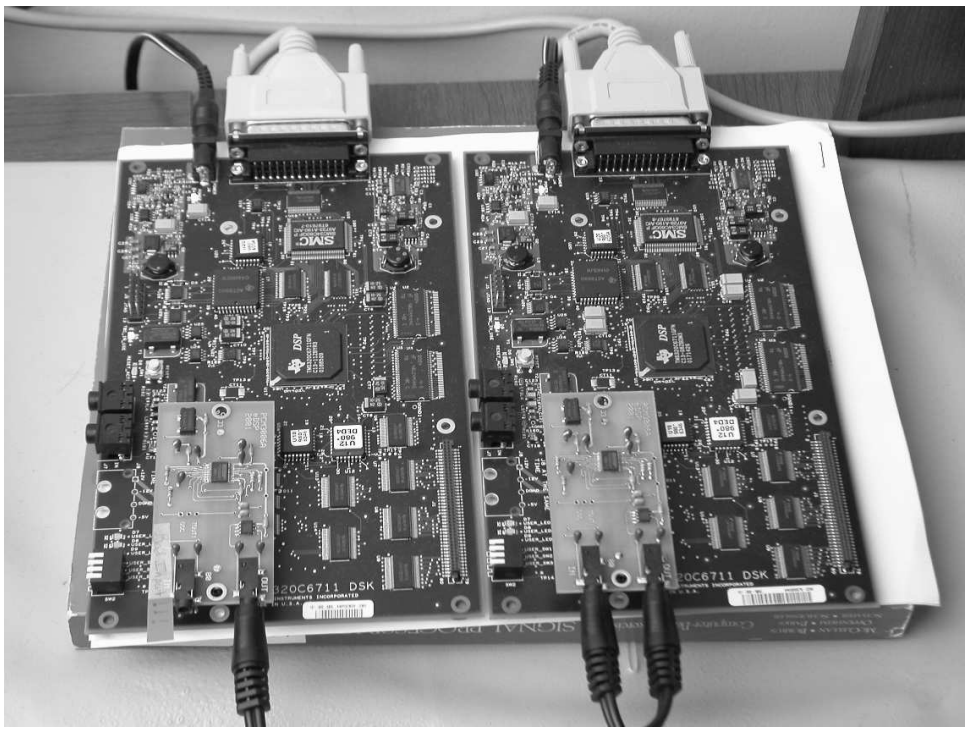

Figure 7.3: 'C6711 DSK transmitter and receiver pair. Notice the two PCM3006 CODEC daughter cards mounted on the lower left corner of the DSKs.

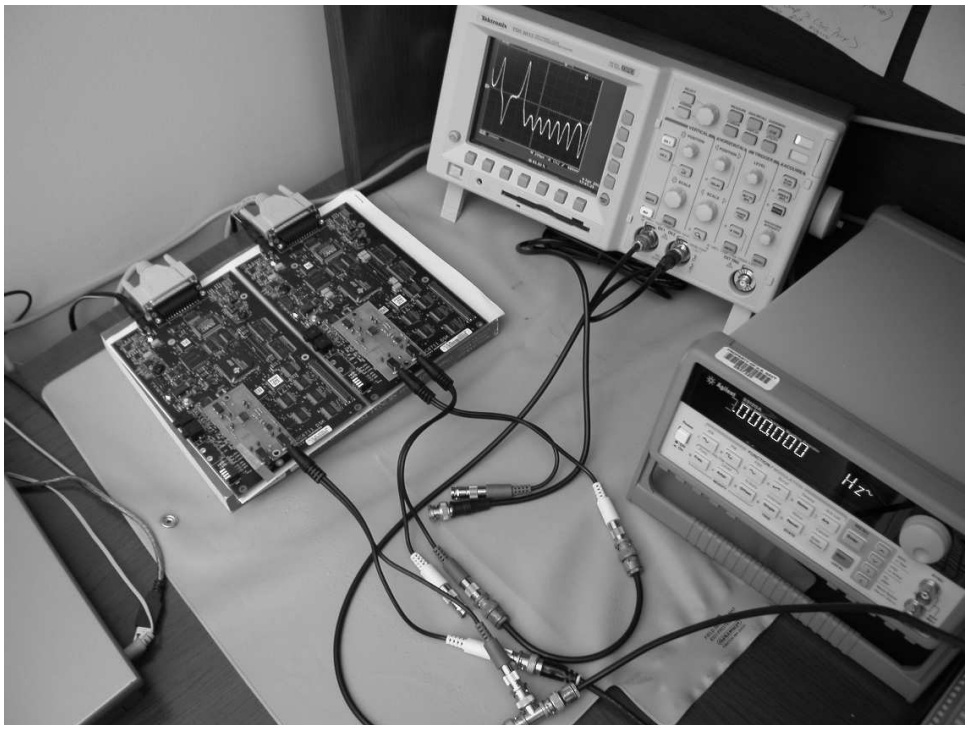

Figure 7.4: 'C6711 DSK transmitter and receiver pair with wires to transfer generated signals from the transmitter to the receiver. One of our oscilloscopes monitors the transmitter waveform. A function generator on the lower left stands by for testing purposes.

Figure 7.5: Hardware development station. Shown here: one of three computers used to conduct simulations and program or control the DSK devices, the two DSK boards with integrated 'C6711 DSPs, two digital dual-channel oscilloscopes, and a Vector Signal Analyzer. The later device measures characteristics of our system in the frequency domain.

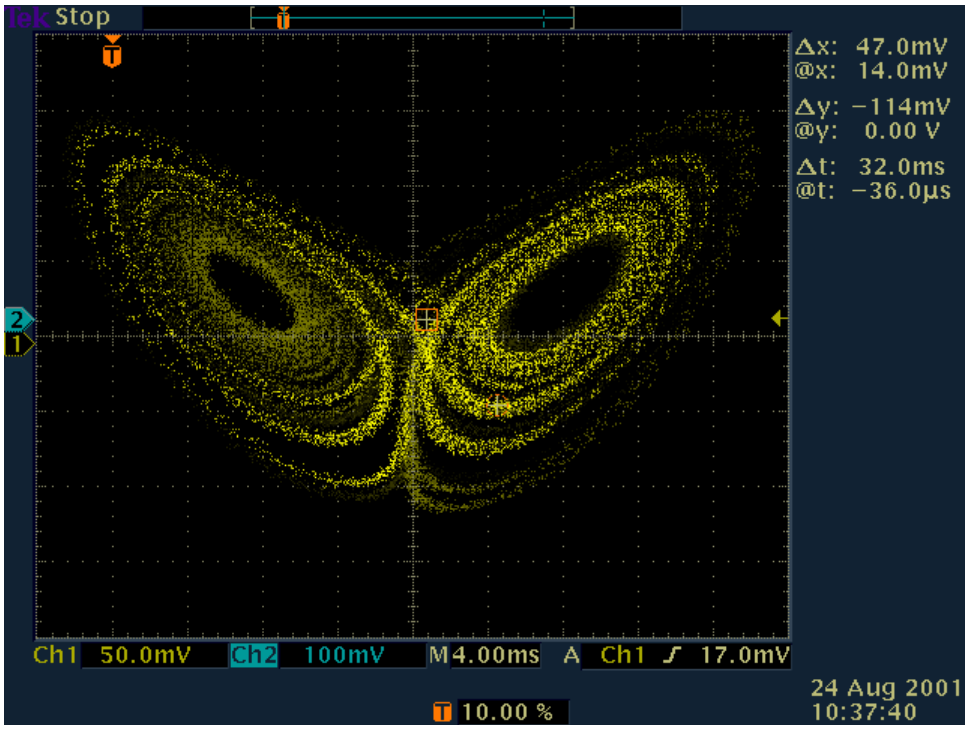

Figure 7.6: Shows phase states u versus v resulting from equation 4.6 for our chaotic transmitter system implemented on DSP hardware. This oscilloscope screen shot depicts our successful implementation of chaos on discrete hardware. The ability to discretely generate a chaotic waveform and generate this type of display was a milestone for our project.

Chapter 8

Bit Error Performance

For any digital communication system, a bit-error probability analysis is one of the most fundamental ways to characterize system performance. Bit-error probability describes the likelihood that a transmitted bit will be received in error at the receiver. Often, the primary reason a receiver will misinterpret a received bit is noise encountered in the communication channel. Due to time constraints, this is the only detrimental effect we will study. During our initial research, we found this type of analysis had been pursued in very few previously published works on chaotic communications systems. As it was initially unclear how robust a communication system based on chaotic parameter modulation would be, the characterization of bit error probability became a significant part of our investigation.

Measuring the bit-error probability of a communications system can be a long and meticulous process. To do these measurements in hardware requires specialized equipment that was not available for this project. Simulating the same property is more accessible, but care must be taken to ensure any results gained will be relatable to real hardware performance. Also, these Monte Carlo simulations tend to be computationally intensive by their nature.

8.1 Monte Carlo Error Counting

The most common method to characterize bit-error probability is by generating bit-error rate (BER) curves. These curves relate bit-error probability to the ratio of bit energy to noise power spectral density, $\frac{E_b}{N_0}$ (measured in dB). For a few specific schemes such as binary phase-shift key (BPSK) a closed form expression for the bit-error rate can be developed. The systems for which closed form expressions exist are the exception. For most systems, results must be obtained from simulations or experimentally comparing bits transmitted to bits received in the hardware systems.

To develop bit-error rate curves through simulation, a full model must be developed for the transmitter, receiver, and communication channel. A large number of bits are sent by the transmitter, in a Monte-Carlo fashion, through the simulated channel and interpreted by the receiver. Bit errors can be counted by comparing the series of bits transmitted to the series received. This ratio is calculated through simulation for several different values of $\frac{E_b}{N_0}$.

8.1.1 Bit Energy and Noise Power Spectral Density

The numerator E_b is the energy transmitted over a bit period. In many communication schemes E_b is constant. In our chaotic communication scheme, it is not. To account for this, we choose to first calculate a reliable average bit energy $E_{b,Avg}$. In general, energy for a continuous time voltage signal $u(t)$ over interval $t_1 \leq t \leq t_2$ is expressed by,

$$\int_{t_1}^{t_2} |u(t)|^2 dt.$$

For a discrete signal $u[n]$ over the time interval $n_1 \leq n \leq n_2$, energy is expressed by,

$$\sum_{n=n_1}^{n_2} |u[n]|^2,$$

and average power is the above expression divided by the number of sample points in the interval.

The denominator N_0 is the noise power spectral density. Power spectral density (PSD) describes the amount of power present at a particular frequency.

Definition 8.1.1. The power spectral density for a deterministic power waveform is

$$\mathcal{P}(f) = \lim_{T\to\infty}(\frac{|W_T(f)|^2}{T}),$$

where $w_T(t)$ and $W_T(f)$ are Fourier Transform pairs and $\mathcal{P}(f)$ has units of Watts per Hertz.[18]

The power spectral density of a Gaussian noise sequence can be found in closed form. It is constant for all frequencies; that is,

$$\mathcal{P}(f) = \frac{N_0}{2}. \tag{8.1}$$

We are able to generate a Gaussian noise sequence to add to our signal during simulation as follows. Choose a particular $\frac{E_b}{N_0}(dB)$ for the bit error probability calculation.

$$\begin{aligned} \frac{E_b}{N_0}(dB) &= 10\log(\frac{E_b}{N_0}), \\ \frac{E_b}{N_0} &= 10^{(\frac{\frac{E_b}{N_0}(dB)}{10})}, \\ N_0 &= \frac{E_{b,Avg}}{\frac{E_b}{N_0}}. \end{aligned} \tag{8.2}$$

Let σ be the standard deviation and σ^2 be the variance of a Gaussian random sequence. The following relationship between noise power spectral density and variance holds:

$$\frac{N_0}{2} = \sigma^2. \tag{8.3}$$

Functions that produce a Gaussian sequence with normal variance ($\sigma^2 = 1$) are common in applications such as MATLAB. To change the variance of a sequence, multiply the sequence by the standard deviation. Let $g[n]$ be a Gaussian distributed sequence with normal variance. Then, $\sigma g[n]$ creates a Gaussian noise sequence with variance σ^2.

We use this property in our bit error probability simulations. The average bit energy $E_{b,avg}$ is held constant while the standard deviation σ is adjusted to achieve the desired ratio $\frac{E_b}{N_0}$.

8.1.2 Bit Error Rate Confidence

The true bit error ratio relates the number of errors encountered in an infinite sequence of bits. If N_b is the number of bits sent and received and n_e is the number of those bits received in error, then the bit error ratio $\bar{n}$ is

$$\bar{n} = \lim_{N_b \to \infty} \frac{n_e}{N_b}. \tag{8.4}$$

In simulation, this limit and thus the ratio $\bar{n}$ can only be approximated by simulating a large number of bits. We would like our calculated ratio to be close to $\bar{n}$ and also to know within what range the real ratio will be.

Our simulation continues transmitting and evaluating bits until 200 error occur. For large total number of bits sent N_b, this number of errors provides more than acceptable confidence in the calculated bit error rate. Confidence is not as good for smaller N_b leading to error rates on the order of 10^{-1}.

8.2 Channel Simulation

Communication channels can be plagued with many different types of interferences. These often vary widely depending many situations including where the transmitter and receiver are physically located, the amount of power being transmitted, and the number of users attempting to use the same and nearby channels.

The most straightforward communication system can be challenged by the following effects in the channel:

1: **Power Attenuation -** It is expected that the receiver will not receive all of the power emitted by the transmitter. For a wired system, losses are due to attenuation in the transmission line. For wireless systems, the problem of power attenuation can be much more severe. Often only a fraction of power radiated by the transmitter actually excites the receiver antenna. Simple attenuation can be corrected with gain at the receiver.

2: **Signal Delay -** The signal cannot propagate instantaneously. Electromagnetic signals propagate through a medium at some speed less than the speed of light. For our system, which is unidirectional and requires no feedback from the receiver, signal delay is not significant to our model.

3: **Phase Delay -** This effect is meaningful only to signals modulated with carrier signals. Some modulation methods require a coherent receiver. This means a sinusoidal oscillator in the receiver must have both the same frequency and phase as a corresponding oscillator in the transmitter.

4: **Gaussian Noise -** Of the four, this is the most important effect on a wireless transmission system. It is expected that the receiver will receive energy other than that which is generated by the transmitter. This excess energy is unwanted and can come in many forms. Usually noise is modeled as an additive process with a Gaussian distribution. This model is referred to as additive white Gaussian noise (AWGN).

Communication channels may include a combination of the following effects that are more difficult to overcome.[19] Our baseband transmission system is subjected to a frequency selective channel.

5: **Nonwhite, Non-Gaussian, and Non-additive Disturbances -** Colored Gaussian noise does not have constant spectral density. This is more realistic than simple AWGN as not all frequencies can pass through a channel with constant attenuation or gain. Interference (unintentional) and jamming (intentional) can be very difficult to overcome. Energy is emitted by another transmitter that overlaps the frequency band used by the communication system. Noise can also be impulsive in nature – occurring relatively infrequently and poorly modeled by a Gaussian process.

6: **Frequency Selective Channels -** These channels do not pass all frequencies with the same level of attenuation. Wires are not generally affected by this effect except at very high frequencies. For wireless systems, atmospheric and ionospheric effects are significant. Also, narrow band antennas may not respond uniformly to the full bandwidth of a transmission. See chapter 10 for a description of our difficulties encountered with a frequency selective channel in our baseband system.

7: **Time-Varying Channels -** These channel effects are most significant for mobile wireless systems. Doppler shift can be significant for a moving receiver or transmitter. For speeds much slower than the speed of light, Doppler shift is proportional to the relative velocity between the transmitter and receiver. Doppler induced fading is of concern for low data rate systems.

Often a transmission travels to a receiver over multiple paths. These multi path channels are especially significant in urban areas where reflections off of structures cause a signal to be received several times with different time delays. Multi path fading is of high concern for high data rate systems.

8.3 Results of Bit Error Probability Simulations

As bit error probability is significant to any communication system, we present results from several different simulations. All bit error probability simulations run in MATLAB with our dual synchronizing receiver scheme.

The first set of simulations highlight the performance of our dual synchronizing receiver scheme with parameter sets that have been used in another published work. Kevin Cuomo, et al. present a communication scheme based on chaos in [7]. Their continuous system is built as an analog circuit. In their single parameter modulation scheme, the β term is the only parameter changed based on the message bit to be transmitted. Cuomo, et al. did not report bit error probability performance for their system. The open squares in Figure 8.1 show the performance of our system when modulating only the β parameter based a message bit. $\beta(0) = 4.0$ and $\beta(1) = 4.4$. The other two parameters are constant: $\sigma = 16.0$ and $\rho = 45.6$.

We utilize the memory of our discrete processor based system in our dual synchronizing receiver scheme. This allows for a more substantial parameter set mismatch between a one-bit and a zero-bit. By running two receivers, one for each parameter set, the transmitter state and receiver state will always reasonably match at the end of any bit period. The asterisks in Figure 8.1 show the performance of our system using a more substantial modulation of β. ($\beta(0) = 4.0$ and $\beta(1) = 4.4$.) The other two parameters are constant: $\sigma = 16.0$ and $\rho = 45.6$.

The open circles in Figure 8.1 show the bit error probability results published in a paper by Carrol and Pecora for comparison.[12] These were the only bit error probability results we could find for a chaotic communication system. We are using a different scheme that we expect to be more covert and our results show better bit error probability performance.

The bit period is a crucial factor for bit error probability simulation because it affects how much time the two receivers have to synchronize to or diverge from the influence signal. This also directly effects system data rate. Our sample count for these simulations is 100 samples per bit. At 48,000 samples per second, the bit period is $T = 2.08msec$. For comparison, the bit period for the open circles is reported to be 200 sec.

8.3.1 Unexpected Source of Errors at High $\frac{E_b}{N_0}$

We have discovered, while trying to ascertain the cause of bit errors, an interesting result. For relatively large ratios of bit energy to noise power spectral density, $\frac{E_b}{N_0}$ (measured in dB), bit errors are largely due to characteristics of the chaotic system instead of noise. For basic transmission schemes like BPSK, the energy in a bit is always the same and errors occur when the noise energy is large. This is not true for our chaotic scheme. Recall that we used average bit energy $E_{b,Avg}$ for our calculations of noise power spectral density.

It turns out that the Lorenz system occasionally goes into regions where the energy of $u(t)$ is significantly less than its average. When the bit window corresponds to these regions, the energy in those bits is smaller than expected. The level of Gaussian noise influence is not adjusted in our simulations to account for lower signal energy. We decided to model system behavior in this way as noise is not something that can usually be adjusted in a

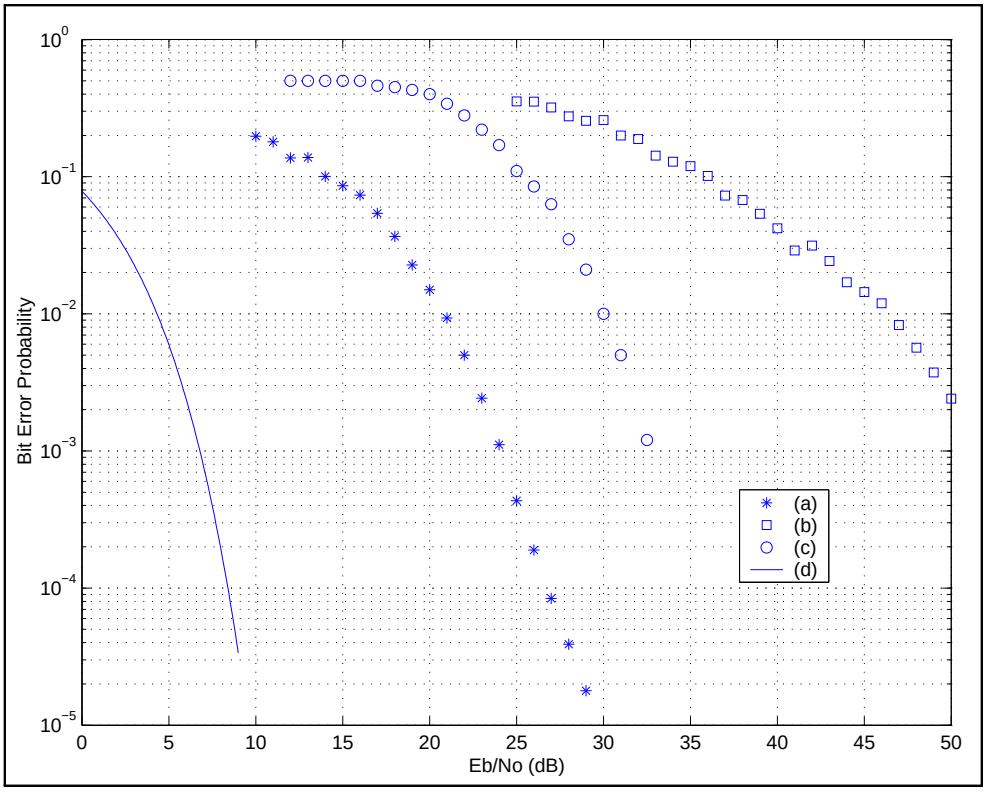

Figure 8.1: Bit error probability as a function of the ratio of energy per bit E_b to noise power spectral density N_0 for several communications schemes. (a), the asterisks show the performance of our discrete system using parameter modulation techniques with a good parameter set (b), the open squares show the performance of our discrete system using the more conservative parameter mismatch used in [7] (c), the open circles show the performance of the multiple attractor system in [12] (d), the solid line shows results for baseband BPSK for comparison.

$\frac{E_b}{N_0}$

$\frac{E_b}{N_0}$ 9

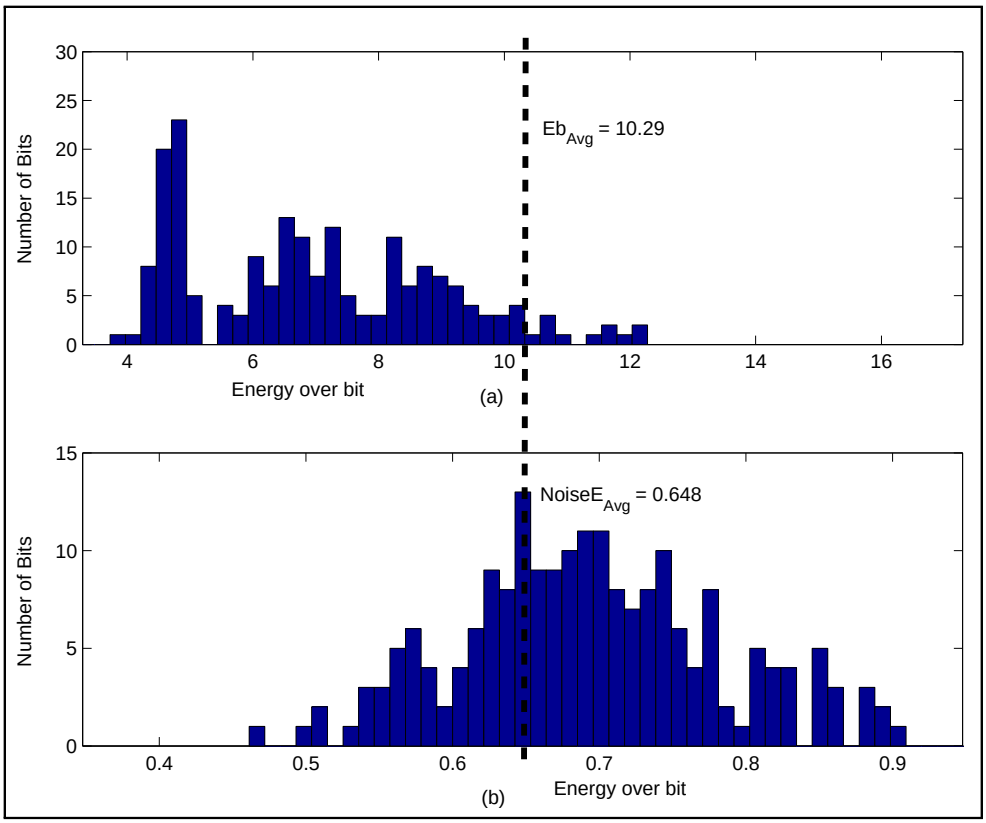

Figure 8.2: Histograms of (a), bit energy and (b), noise energy for the dual synchronizing response system over 200 errors bits at $\frac{E_b}{N_0} = 29dB$. The average bit energy and average noise energy for all bits is indicated by the dashed line.

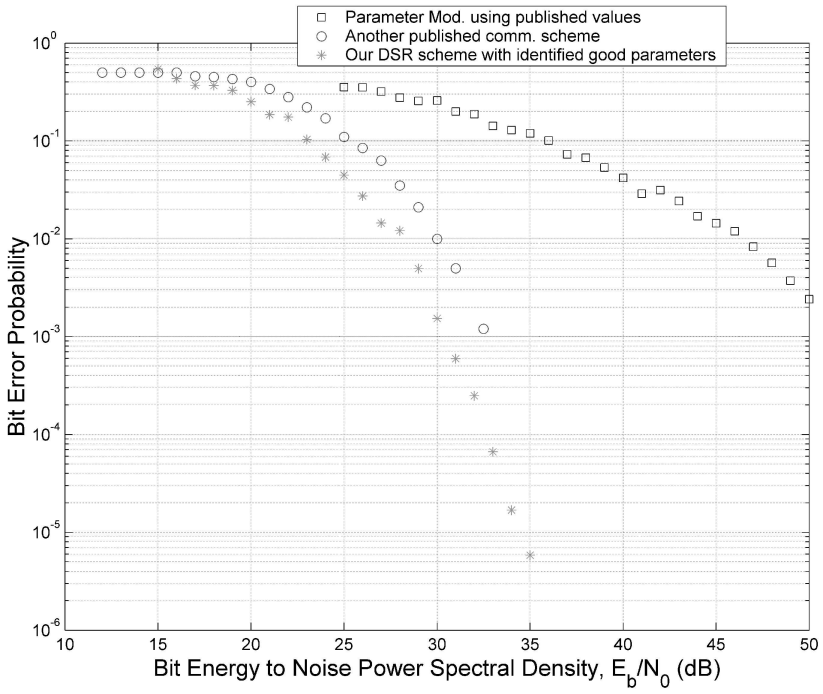

Figure 8.3: The parameter sets identified using our framework including amplitude scaling lead to the bit error probability curve indicated by the asterisks. In comparison with the performance of our system without scaling in figure 8.1, we see that it will take more bit energy to achieve the same bit error probability with scaling.

Chapter 9

Characterizing Frequency Domain Properties

Often it is useful to characterize a signal using more than just time-domain observations using an oscilloscope. A spectrum analyzer is a specialized piece of equipment that measures the distribution of signal power as a function of frequency. This type of analysis is used heavily in electronic warfare. In support of our goal to evaluate this system's applicability for covert communication, we measure some frequency domain characteristics of our system.

9.1 Signal Camouflage

Initially, our only capability to look at our system's performance in the frequency domain was by MATLAB simulations [20]. We now have a hardware system that can be analyzed using signal analysis equipment. The primary device for doing this is a sweeping spectrum analyzer. This system searches in a sweep pattern through many frequencies to measure the amount of energy being received at each particular frequency. This is one method by which a transmission might be detected by an electronic warfare platform. Figure 9.1 shows some results of MATLAB simulations comparing the frequency response of our chaotic transmitter with the noise floor. The fact that our transmitter is below the noise floor is good for covertness.

Results depicted in figure 9.1 are very promising for applications of chaos to covert military communication. The system appears able to spread its energy in such a way that simple observation of power in the frequency domain will not detect the signal. We next present some hardware observations that support this indication.

9.2 Frequency Domain Measurements in Hardware

Using our sweeping spectrum analyzer, we measure in hardware the distribution of energy in the frequency domain. We confirm that the random-like behavior initially observed in

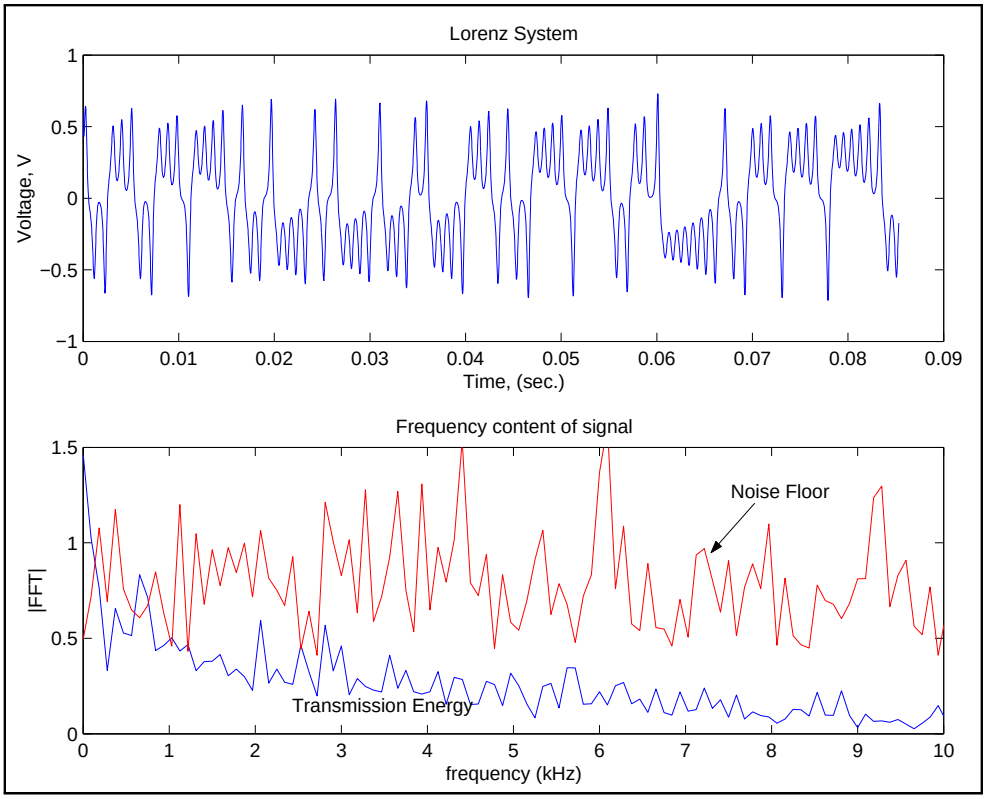

Figure 9.1: The above look at the frequency content of our signal indicates signal energy is spread under the noise floor. The ratio of bit energy to noise power spectral density is $\frac{E_b}{N_0} = 25dB$. Reading from figure 8.1 shows that about one in every two thousand bits received are in error.

chaotic systems does in fact lead to smooth and wide energy distribution in the frequency domain.

Figure 9.2 shows the energy of our transmitted signal observed by the spectrum analyzer over one sweep. The energy distribution of this figure changes substantially from sweep to sweep of the equipment, just like it would for a random signal.

Figure 9.3 is same spectrum with video averaging turned on and set to ten averages. This allows us to see the general shape of the energy distribution associated with this chaotic system. It is promising that the spectrum is relatively smooth without any distinguishing peaks of energy.

While likely too small to be of concern for our covertness investigation, we take interest in the two notches that appear and remain in the averaged system spectrum. These are shown in figure 9.3 at approximately 6kHz and 12kHz. It turns out that when our transmitter actually begins modulating its parameter set to send a message, the small notches are washed out in the same amount of averaging. This is depicted in figure 9.4.

In addition to the basic frequency spectrum snapshots and averages depicted in figures 9.2 through 9.4, we use spectrograms to analyze our systems covert performance. Spectrograms allow a hybrid analysis. Both time domain and frequency domain information can be

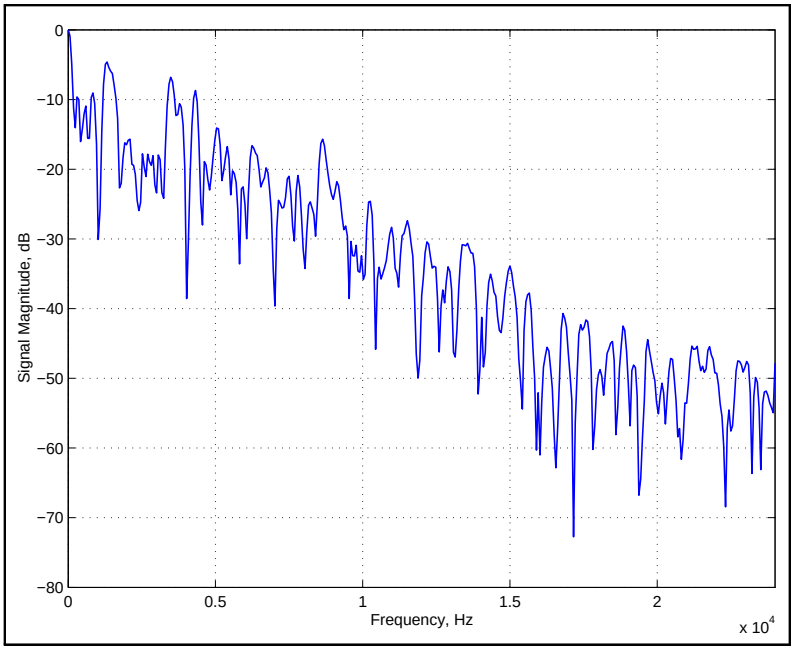

Figure 9.2: Frequency spectrum of Lorenz system observed on spectrum analyzer without averaging.

5

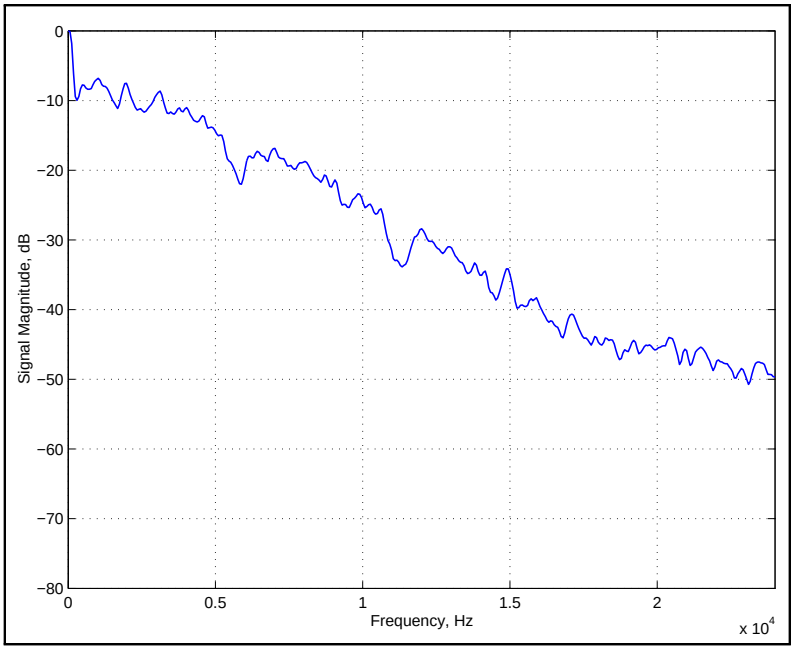

Figure 9.3: Frequency spectrum of Lorenz system observed on spectrum analyzer with ten video averages. Two notches in the otherwise smooth falloff spectrum remain for longer averages as well. (6kHz and 12kHz)

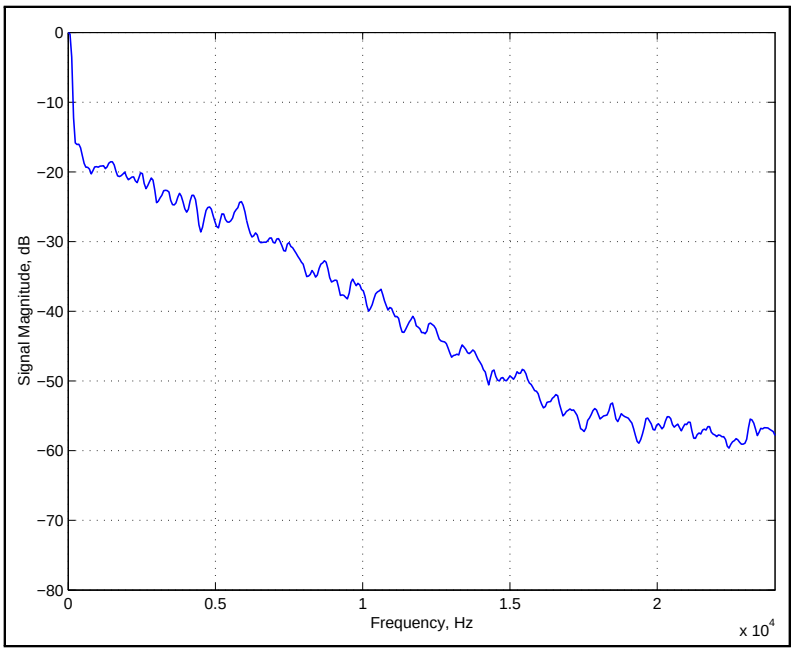

Figure 9.4: Frequency spectrum of the Lorenz system with ten video averages. Every 100 samples, the value of β is toggled between 4.0 and 2.5. The resulting spectrum washes out the characteristic notches seen when the parameter set is held constant. Without averaging, it is not possible to distinguish the spectrums of the switching signal, or either constant parameter system.

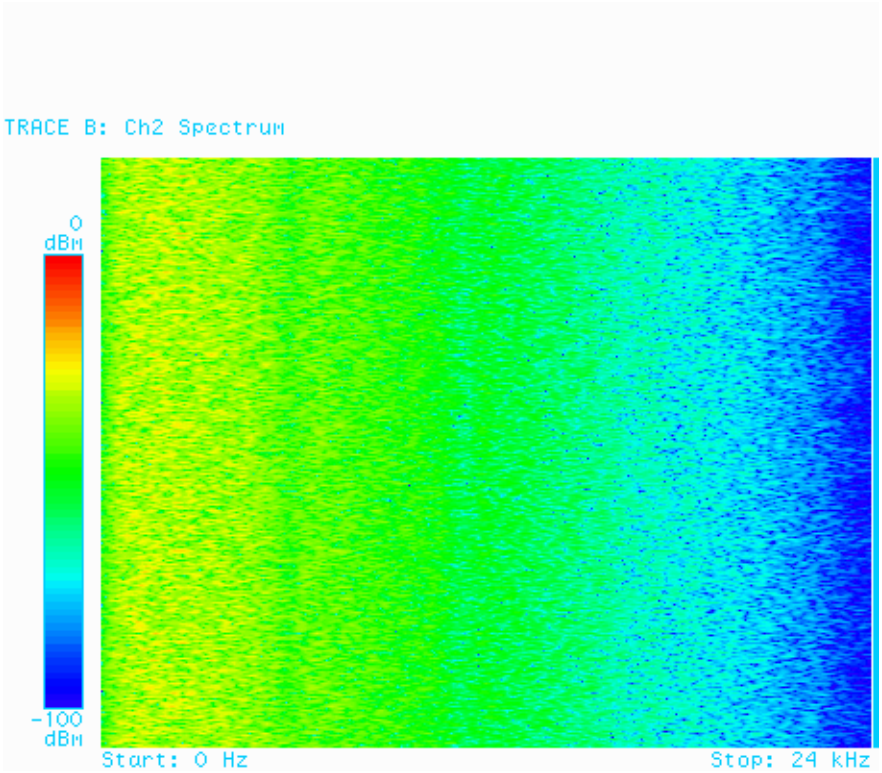

Figure 9.5: This spectrogram characterizes the $x(t)$ signal from the Lorenz system operating with a constant parameter set $\sigma = 16$, $\rho = 45.6$, and $\beta = 4$. The smooth energy distribution that remains relatively constant over time is very encouraging for system covertness.

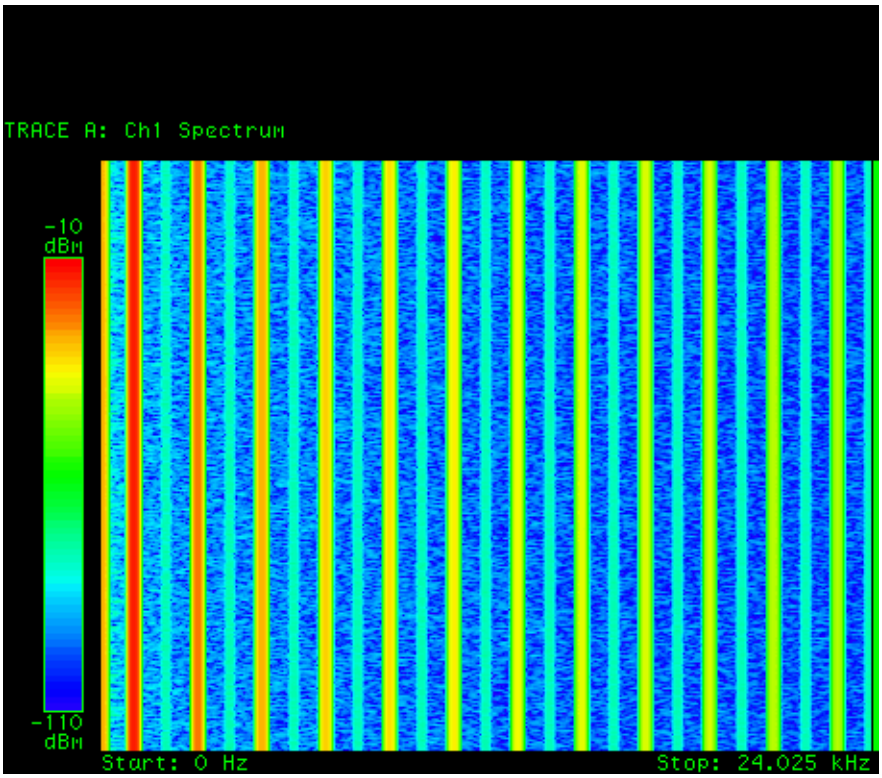

Figure 9.6: This spectrogram characterizes a square wave for comparison. Periodic signals are much more easily detected than signals with smooth spectrums as seen in figure 9.5.

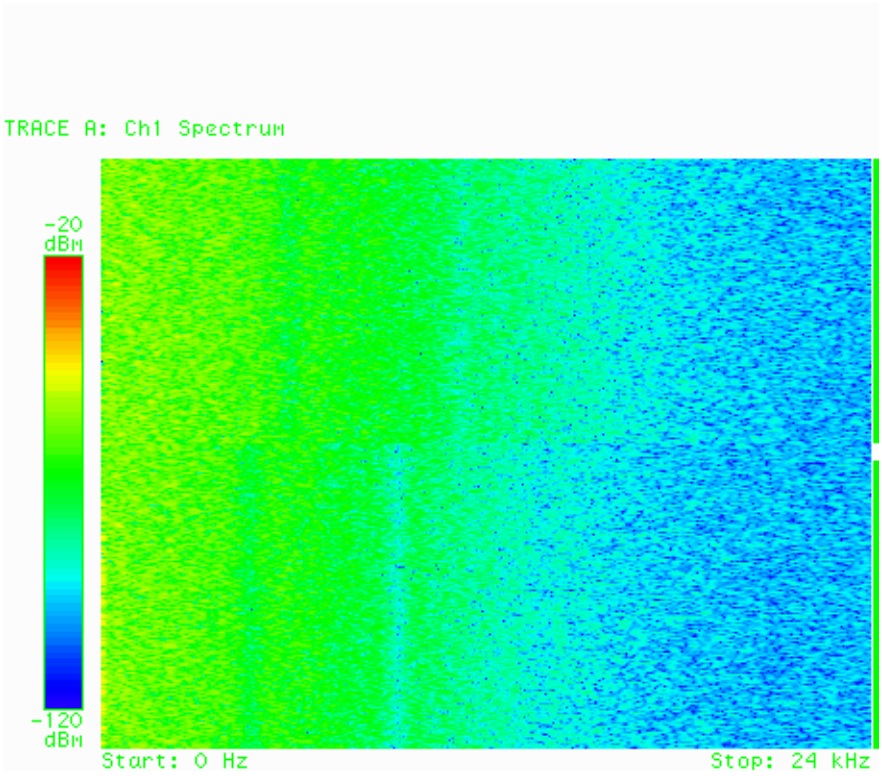

Figure 9.7: Half-way though the observation of this spectrogram, the β parameter is changed from 4.0 to 2.5. It is difficult, but one can see a slight change in the signal spectrum. This spectrogram is only for discussion, the switching scheme in figure 9.8 is more realistic for our parameter modulation scheme.

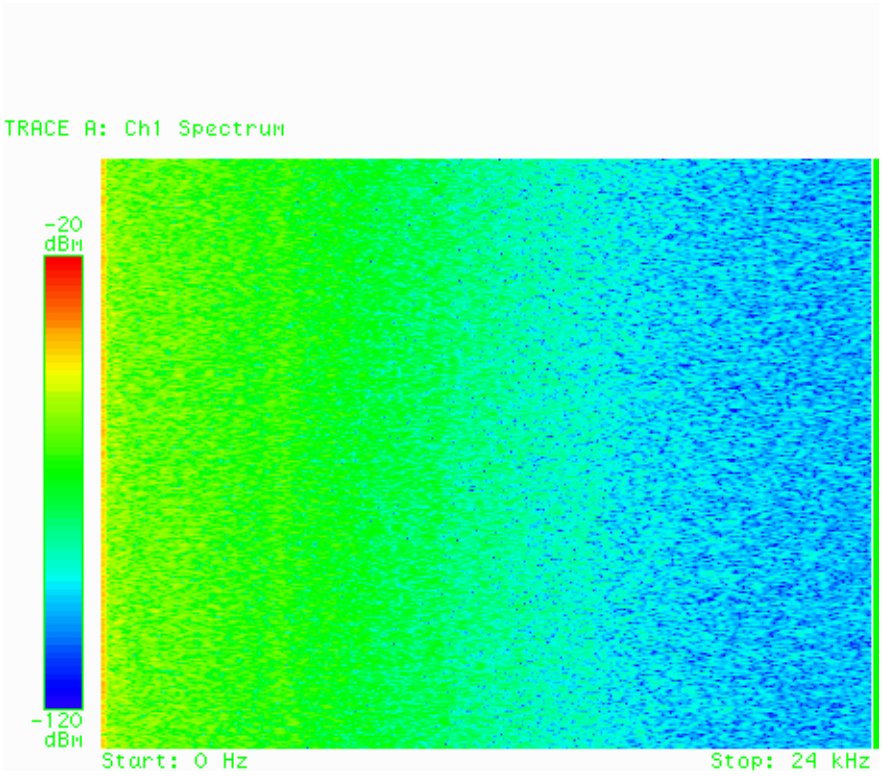

Figure 9.8: Over the observation of this spectrogram, the β parameter of the Lorenz system is toggled between 4.0 and 2.5 every 100 samples. The results is that the two spectrums for the constant parameter systems get blurred into this spectrogram. It would not be possible to ascertain β values after the fact based on this spectrogram data.

Chapter 10

Low Frequency Attenuation Challenges

One significant milestone for implementing our chaotic communication system in hardware was to demonstrate the ability to synchronize a transmitter and receiver. This was a important milestone in our project that had to be achieved before any message could be sent and recovered. We encountered an unexpected amount of difficulty achieving synchronization.

In our work to identify the problem, we discovered some limitations of the Lorenz System. If not planned for, these limitations could be detrimental to a real system even though they are difficult to pick up in simulation alone.

We find that the Lorenz System's ability to synchronize is limited or even completely hampered if the influence is sent through a frequency selective channel. We were not trying to send the signal through such a channel, but realized our CODECs have a high-pass filter fixed to the analog-to-digital converter that cannot be bypassed. Our CODECs were designed to be used in audio-frequency applications. Frequencies near DC are not significant for audio. We find the attenuated frequencies are significant to our system for synchronization.

To test our hypothesis we increased the time scaling factor T_S as much as possible. Increasing T_S effectively stretches the signal bandwidth away from 0 Hz. Since the same amount of power is transmitted, less power is concentrated in the frequency range that the CODECs attenuate. The time scaling factor can only be increased limitedly. The upper end of the signal bandwidth must remain below the Nyquist sampling rate for our CODECs (24 kHz).

Making this adjustment to our system, the systems synchronize much better. Perfect synchronization is still not achieved, so we seek to determine if the remaining signal attenuation explained the problem. The specifications for our CODEC hardware describe the characteristics of the high-pass filter in the analog-to-digital converter stage. Figure 10.1 shows the reported attenuation versus frequency of this component[17].

We use a vector signal analyzer to treat the talkthrough system as a filter and measure its frequency response. Figures 10.2 and 10.3 show the resulting magnitude and phase response of the talkthrough system for two different frequency ranges.

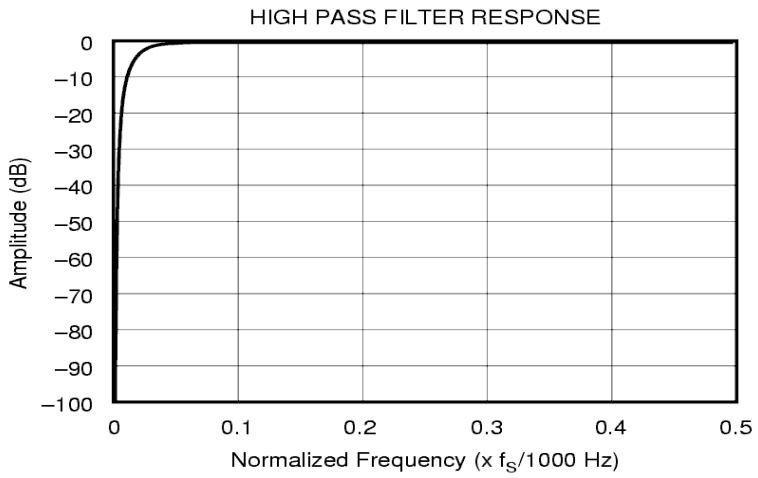

Figure 10.1: Low Frequency Response of ADC Digital Filter reported by PCM3006 Specifications Sheet. The curve relates signal attenuation (in dB) to normalized frequency. On the horizontal axis, 0.1 is equivalent to 4.8 Hz.

10.1 Near-DC Attenuation Model

6

10.2 Hardware System Attenuation

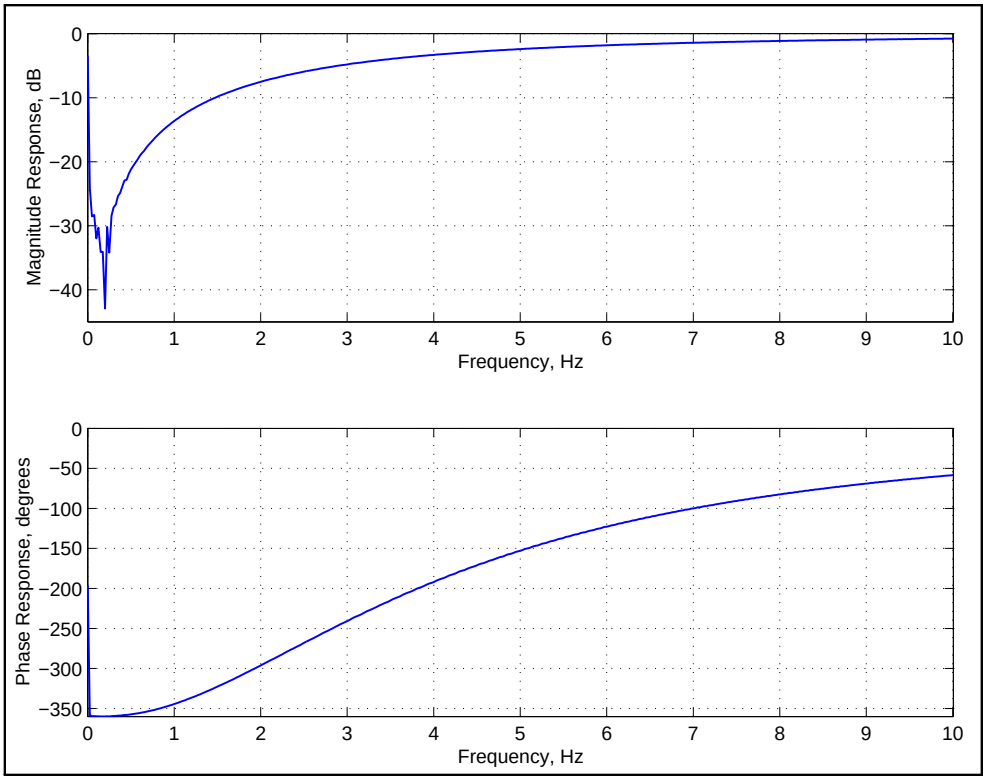

Figure 10.2: Observed magnitude and phase response of the hardware talkthrough system described. Frequency ranges between 0 and 10 Hz in order to exam filter attenuation near DC. Response is similar in form to the published filter response in the CODEC data sheet.

section must be compensated for. Without any compensation, even a simple talkthrough function produces results as seen in figure 10.7.

This time delay (about $640\mu s$) is a combination of input sampling time, output conversion time, CODEC filter delay, and processing delays in the DSP. To perform a simple talkthrough function (pass input to output) requires very little processing and so delays caused by the DSP are insignificant. For more advance systems that require data manipulation (such as our transmitter and receiver system which require calculating one more steps via the Runge-Kutta 4-5 Algorithm) the most serious time limitation is that all processing for one sample must be completed before the next sample arrives. Processor capability (in MFLOPS or MIPS) must be matched with CODEC sampling rate and algorithm complexity in order to ensure a system meets the timing requirements.

As a worst case, delay associated with input sampling and output conversion total twice the sampling period of the CODEC. For the PCM3006, which samples at 48 kHz, $2T_s = \frac{2}{48000} = 41.67\mu s$. Much more significant are the delays caused by the digital filters incorporated in the A/D and D/A converters. As reported in the data sheet, these delays are $362.50\mu s$ and $231.25\mu s$, respectfully. The sum of these delays closely agrees with our observation for the talkthrough application.

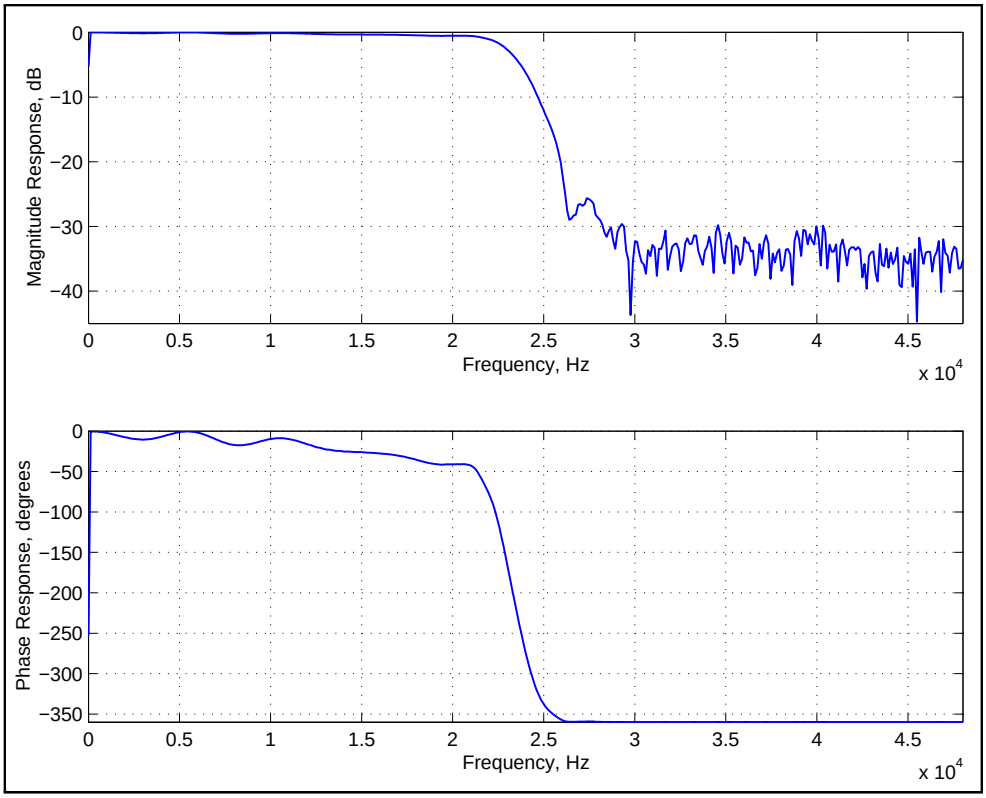

Figure 10.3: Observed magnitude and phase response of the hardware talk-through system described. Frequency ranges between 0 and 48 kHz in order to find response to all frequencies up to the sampling frequency. The effect of a low pass filter with cutoff at $\frac{f_s}{2} = 24kHz$ is apparent.

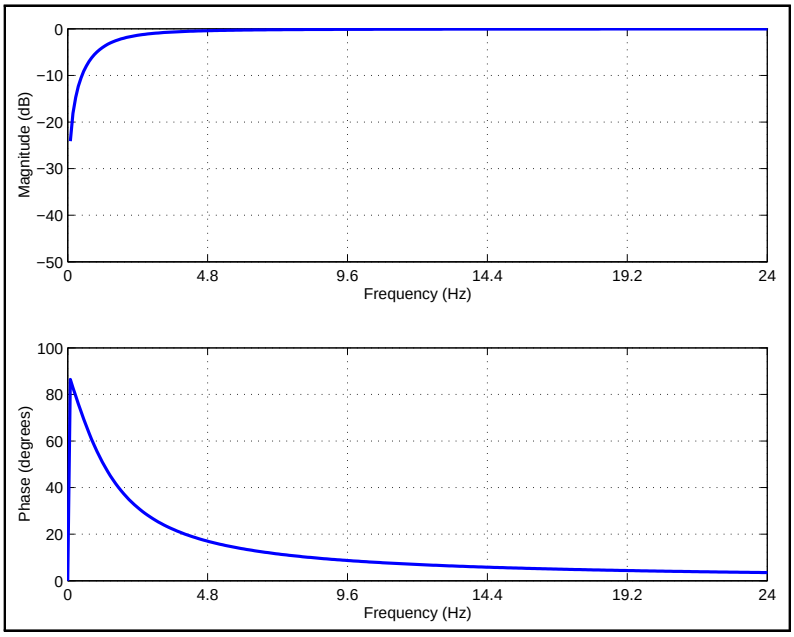

Figure 10.4: Magnitude and Phase Response of our first order filter to model response reported by CODEC data sheet.

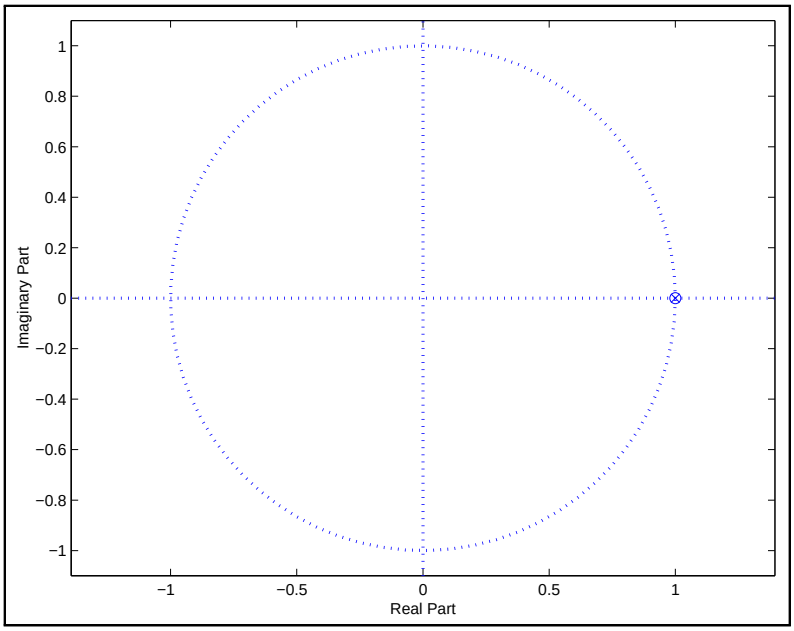

Figure 10.5: Z-plane representation of our high pass filter model. Single zeros is located very close to the unit circle in order to simulate steep roll off near DC.

Figure 10.6: Synchronization simulation in MATLAB using the low frequency attenuation model described. Synchronization error is similar to our hardware observations.

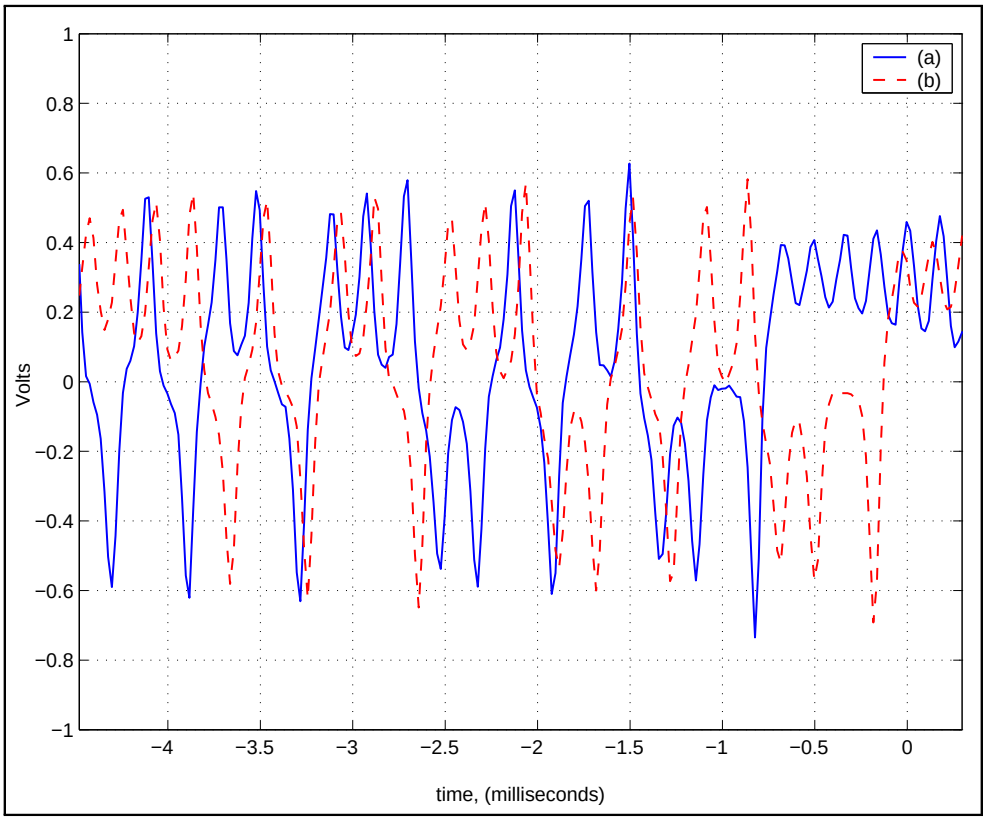

Figure 10.7: Operation of basic talkthrough function observed on a two channel spectrum analyzer. Due to time delays related to sampling and processing, a comparison is difficult to make visually. (a), The solid line indicates the input to the CODEC and, (b) the dashed line indicates the talkthrough output.

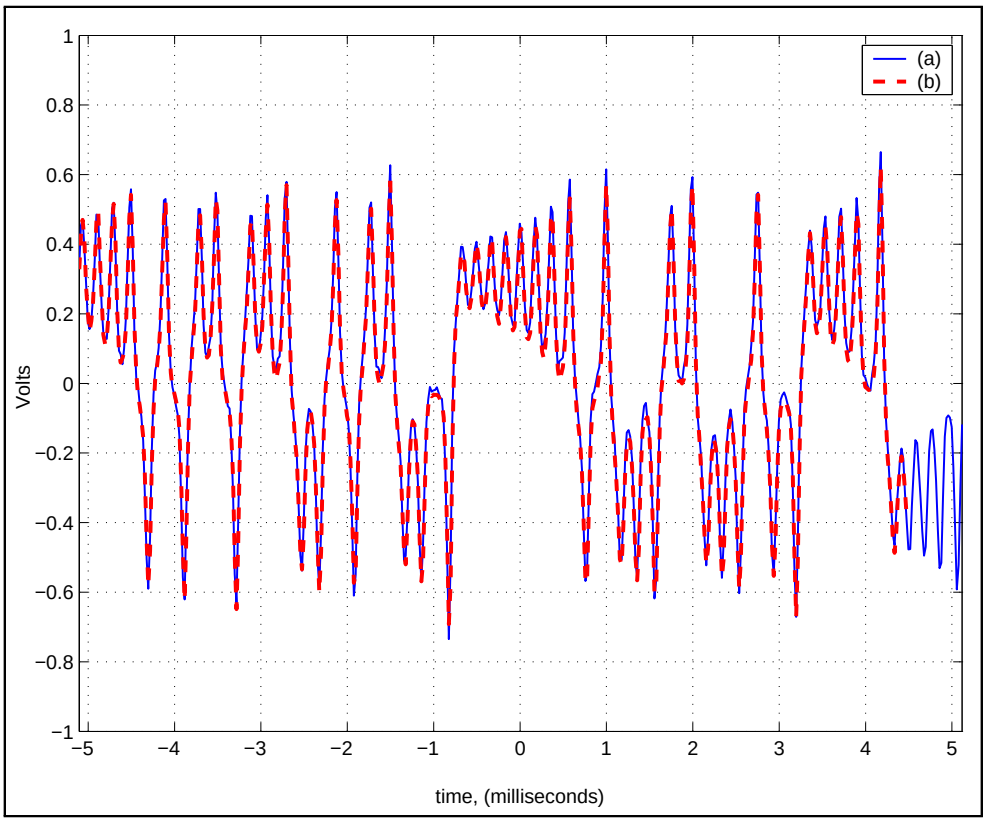

Figure 10.8: Output of talkthrough system is shifted earlier in time to facilitate comparison of the signals observed in figure 10.7. (a), The solid line indicates the input to the CODEC and, (b) the dashed line indicates the talkthrough output.

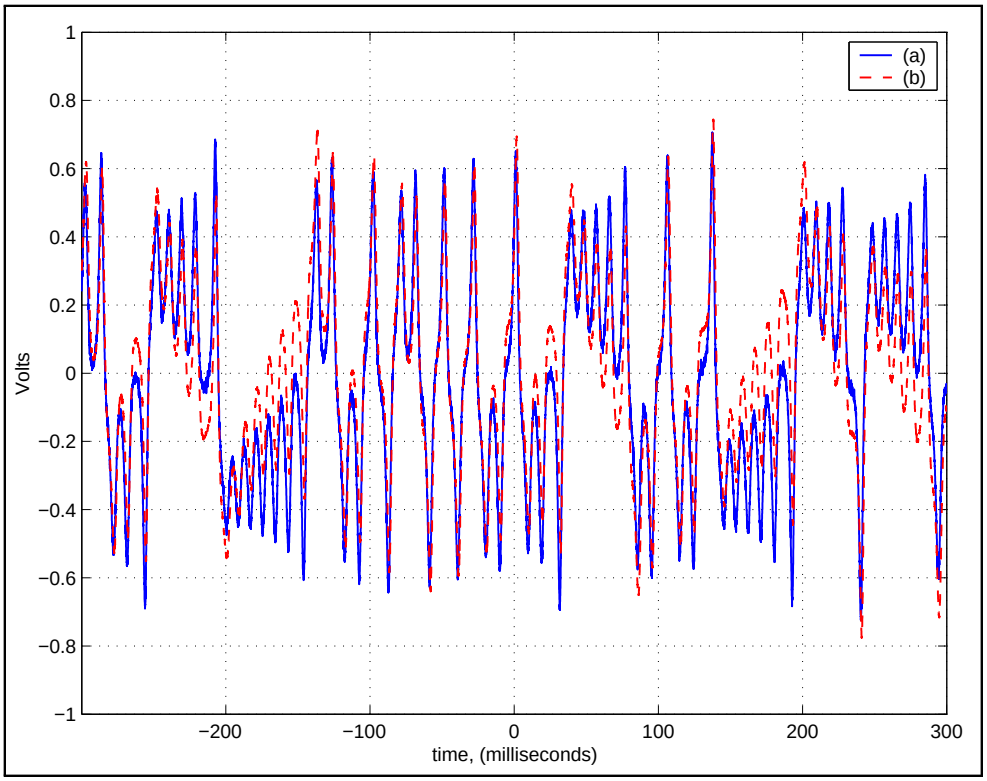

Figure 10.9: Talkthrough response as in figure 10.7, except that the input Lorenz System is time scaled to one-twentieth the first rate. More signal energy is near DC and so attenuation causes more error between the input and output of the talkthrough system. (a), The solid line indicates the input to the CODEC and, (b) the dashed line indicates the talkthrough output.

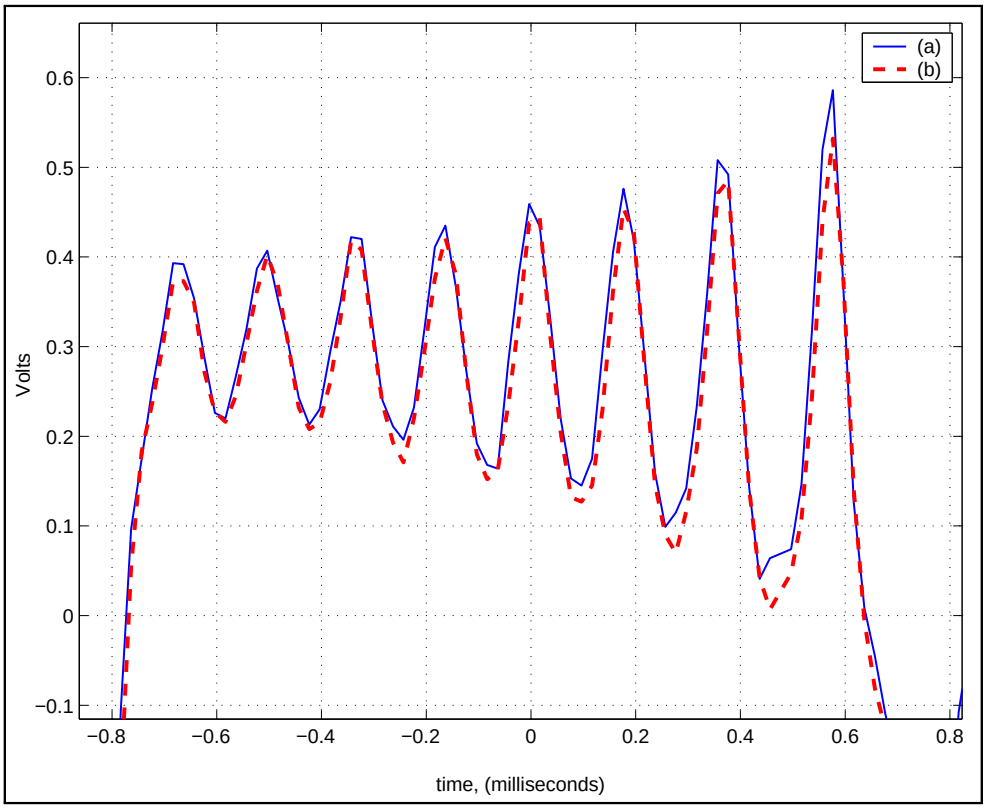

Figure 10.10: View of smaller region in figure 10.7. In this region, the affect of low-frequency attenuation is most obvious. (a), The solid line indicates the input to the CODEC and, (b) the dashed line indicates the talkthrough output.

Chapter 11

Conclusion

We have presented several promising results, including the development of our dual synchronizing receiver method and a framework for selecting good parameter sets for a parameter modulation scheme. Several unexpected setbacks with our hardware, namely the near-DC attenuation of the CODECs, made progress towards a hardware implemented communication system very difficult. Recently we have begun to make our first developments in this area, and expect to move quickly now that problem areas have been identified.

11.1 Chaotic Communication System Partially Implemented

Figure 11.1 shows real bit-sequence recovery achieved in hardware. This result does not use our dual synchronizing receiver scheme. We expect to implement that shortly.

Instead, the transmitter toggles between two parameter sets based on the message bit and the parameter set of the receiver remains the same. Error between what the receiver expects to receive every next channel and what is actually received is compared. When the parameter sets are matched, the error should be near zero. A mismatch should lead to large error. Figure 11.1 shows a good bit sequence. On the other hand, figure 11.2, shows the problem that arrises due to synchronization problems and noise.

11.2 Our Overall Parameter Selection Method

Figure 11.3 shows the overall procedure we use to select a good pair of parameter sets for our dual synchronizing receiver scheme.

The bit error probability simulations discussed in chapter 8 confirm that this framework does indeed work to significantly improve the performance of our system. The next improvement to make would be to implement the time scaling discussed as well. This would help ensure consistent frequency domain properties for both parameter sets.

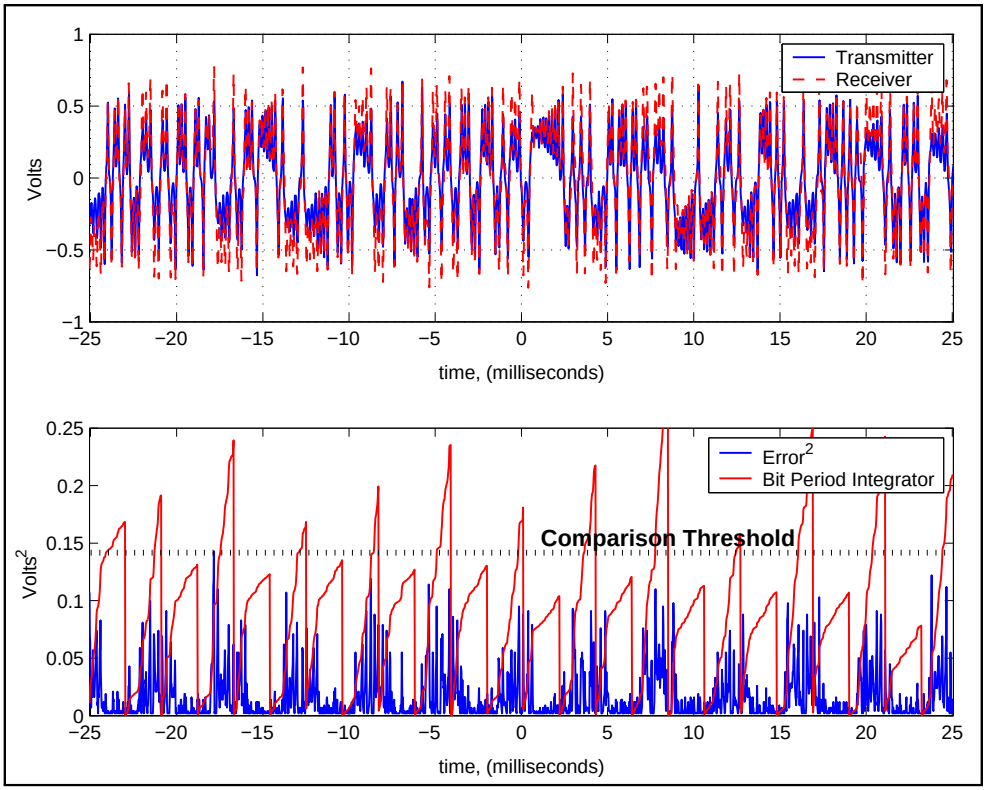

Figure 11.1: Our first received bits in hardware using chaos. Without noise, there is a reasonable expectation that the message will be recovered.

11.3 Problems for the future

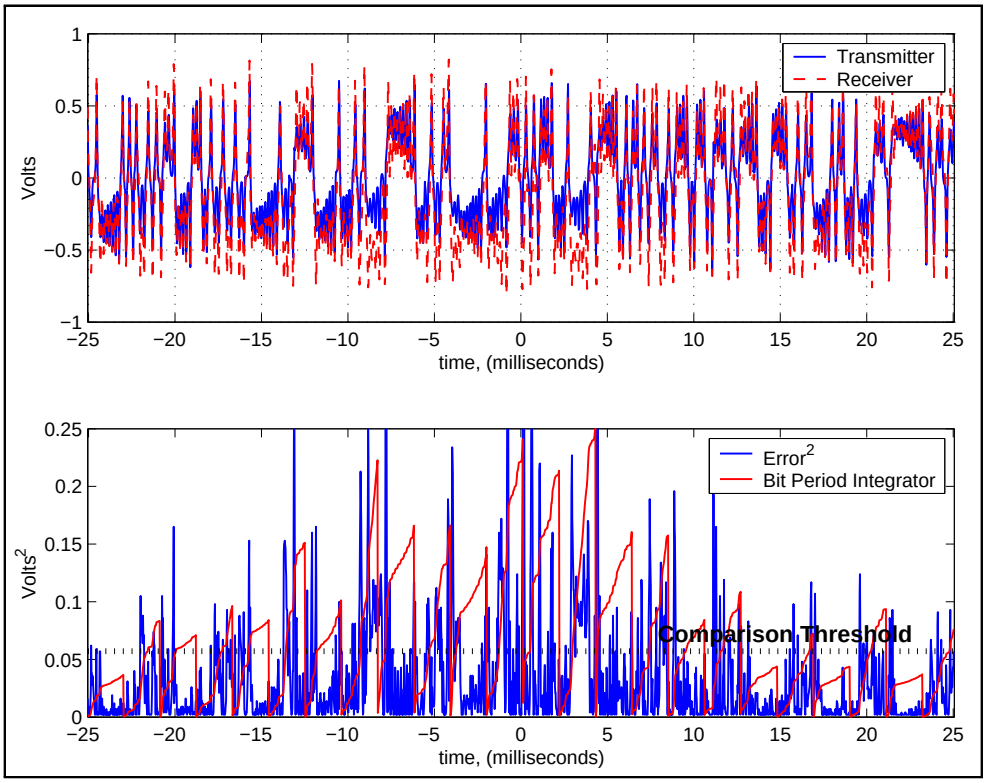

Figure 11.2: Over this time window, synchronization fails for the same reasons we have presented. Several bits are received in error. As the system is not being influence by any extra noise, the synchronization problem is attributed to the near-DC attenuation of our CODECs.

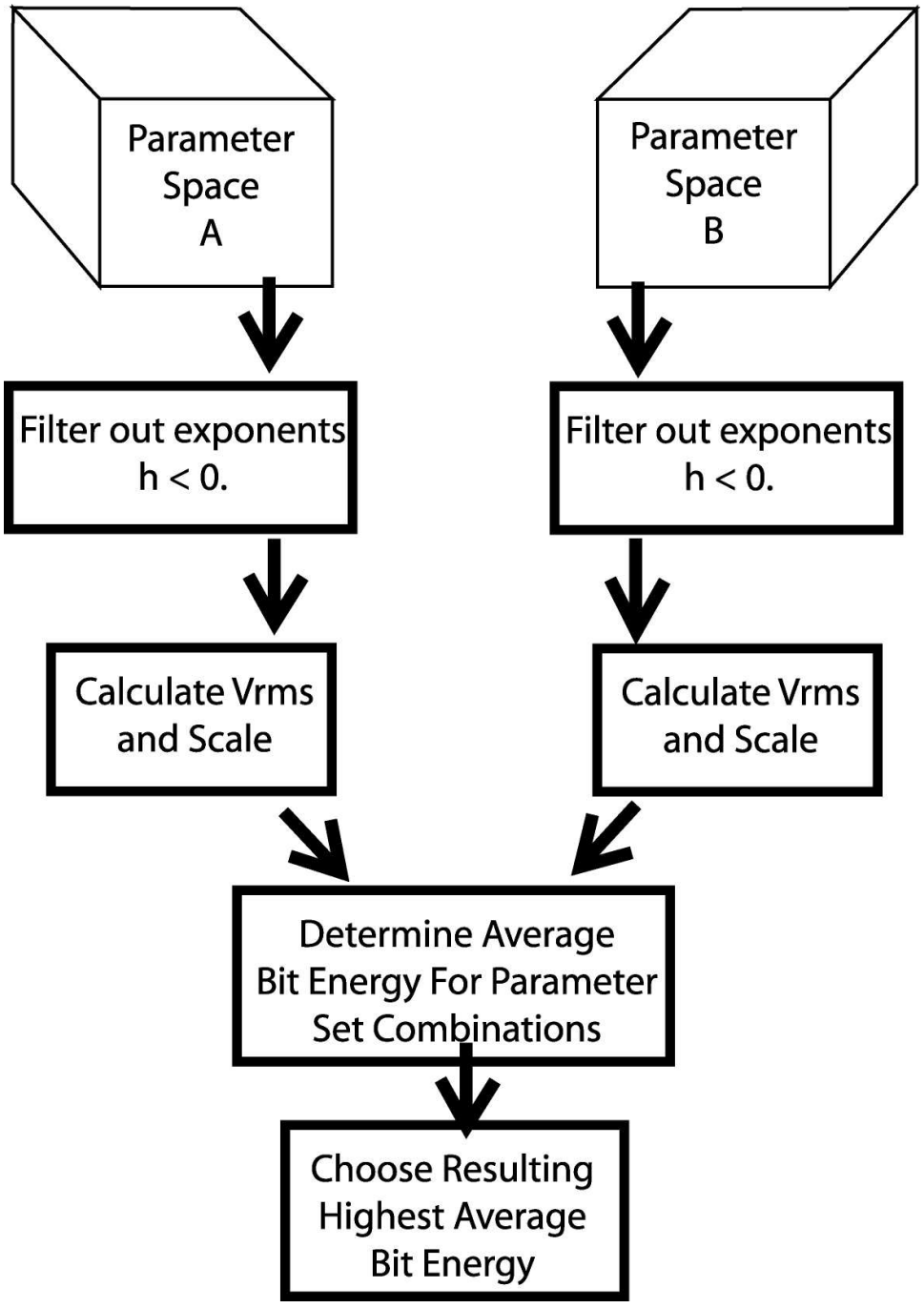

Figure 11.3: Our developed framework for improving the performance of our system while emphasizing covertness.

Bibliography

[1] J. Gleick, *Chaos: Making A New Science*, Penguin Books, New York, 1988.

[2] Army Research Office, "Aro program update 2001-1: New mathematical foundations program replaces applied analysis program," Forwarded email to author, January 2001.

[3] T.Y. Li and J.A. Yorke, "Period three implies chaos," *Amer. Math. Monthly*, vol. 82, pp. 985, 1975.

[4] S.H. Strogatz, *Nonlinear Dynamics and Chaos*, Addison-Wesley, Reading, MA, 1994.

[5] K.T. Alligood, T.D. Sauer, and J.A. Yorke, *Chaos: An Introduction to Dynamical Systems*, Springer-Verlag, New York, 1996.

[6] C. Sparrow, *The Lorenz Equations: Bifurcations, Chaos, and Strange Attractors*, Springer-Verlag, New York, 1982.

[7] K.M. Cuomo, A.V. Oppenheim, and S.H. Strogatz, "Synchronization of lorenz-based chaotic circuits with applications to communications," *IEEE Trans. on Circuits and Systems*, vol. 40, pp. 626–632, Oct. 1993.

[8] M. Itoh, "Spread spectrum communication via chaos," *International Journal of Bifurcation and Chaos*, vol. 9, no. 1, pp. 155–213, 1999.

[9] E.M. Bollt and Y.C. Lai, "Dynamics of coding in communicating with chaos," *Physical Review E*, vol. 58, no. 2, pp. 1724–1736, Aug. 1998.

[10] A. He, K. Li, L. Yang, and Y. Shi, "Tdma secure communication scheme based on synchronization of chua's circuits," *Journal of Circuits, Systems, and Computers*, vol. 10, no. 3&4, pp. 147–158, 2000.

[11] L.M. Pecora et al., "Fundamentals of synchronization in chaotic systems, concepts, and applications," *Chaos*, vol. 7, pp. 520–533, 1990.

[12] T.L. Carroll and L.M. Pecora, "Using multiple attractor chaotic systems for communication," *Chaos*, vol. 9, pp. 445–451, June 1999.

[13] L.M. Pecora and T.L. Carroll, "Synchronization in chaotic systems," *Phys. Rev. Lett.*, vol. 64, pp. 821–824, Feb. 1990.

[14] A.V. Oppenheim and A.S. Willsky, *Signals and Systems*, Prentice Hall, New Jersey, second edition, 1997.

[15] H.K. Khalil, *Nonlinear Systems*, Prentice Hall, New York, second edition, 1996.

[16] L. Dieci, R.D. Russell, and E.S. Vleck, "On the computation of lyapunov exponents for continuous dynamical systems," *SIAM J. Numerical Analysis*, vol. 34, no. 1, pp. 402–423, Feb. 1997.

[17] Burr Brown Inc., "Pcm3006 data sheet," February 1998.

[18] L.W. Couch, *Digital and Analog Communication Systems*, Prentice Hall, New Jersey, sixth edition, 2000.

[19] F.M. Gardner and J.D. Baker, *Simulation Techniques: Models of Communication Signals and Processes*, Wiley-Interscience, New York, 1997.

[20] The MathWorks Inc., "Matlab, release 12.1," May 2001.

[21] *IEEE Standard for Binary Floating-Point Arithmetic*, Institute of Electrical and Electronics Engineers, August 1985.

[22] R.L. Burden, J.D. Faires, and A.C. Reynolds, *Numerical Analysis*, Prindle, Weber & Schmidt, Boston, 1978.

[23] S.L. Hahn, *Hilbert Transforms in Signal Processing*, Artech House, Boston, 1996.

[24] S.A. Tretter, *Communication System Design Using DSP Algorithms*, Plenum Press, New York, 1995.

Appendix A

Numerically Solving Differential Equations

The Lorenz system and many other chaotic systems that we might like to utilize in our chaotic communication scheme are continuous in time and can take on an infinite number of states. Neither of these two properties can be preserved in a discrete system such as the computers we use for simulation and our digital signal processors that we use to implement the communication scheme. Three limitations exist.

1: **Discrete State Representation -** Digital devices such as microprocessors are not able to operate on or store an infinite number of values. Instead they are limited to a fixed number of values depending on how many pieces of memory or *bits* are allocated to each number. In our work, three different storage classes are used. All of them turn out to be less limiting than the limitations described in numbers 2 and 3 below.

   - **Computers -** All results of simulations and other calculations presented in this project utilize the 64-bit double-precision floating-point standard to represent fractional numbers. Established by the Institute of Electrical and Electronics Engineers (IEEE), this standard assigns one sign bit, fifty two bits for the mantissa, and eleven bits for an exponent [21]. Even if our exponent never changes in our calculations the mantissa alone can represent 4503599627370496 different numbers.
   - **Digital Signal Processors -** For our hardware system, the digital signal processors are floating-point devices. More typically DSPs are fixed point and can only represent integers due to the added cost required to incorporate floating point functional units. Such devices can perform calculations faster and are easier to produce. Floating-point capability simplifies the development process because we do not have to worry about scaling our data. The floating-point standard is also defined by the IEEE [21]. One bit represents the sign of the number, twenty three represent the mantissa, and eight represent the exponent. As the names imply, the floating-point standard can

represent half the number of values as double-precision floating-point.

- **CODECs -** In terms of representing the state of our chaotic system, the analog-to-digital converter and digital-to-analog converter (CODEC collectively) introduce the greatest representation limitation of the three discrete systems described. We use 16-bit fixed-point CODECs. This means that they can sample and output 2^{16} or 65536 different values. The devices operate in the range -1 to +1 Volts so the smallest gradation is $\frac{2\ Volts}{65536\ steps} = 30.52\ \frac{\mu V}{step}$. At this scale, noise in the electronics is significant. Given a choice, it would be more productive to tackle the limitations of numbers 2 and 3 below first.

2: Discrete Time and Real Time Considerations - Two limitations in regard to time exist. First, just like discrete state representation, the system state can only be represented at discrete times. For example, we might be able to calculate the state of the system at 1.000 sec. and 1.001 sec., but cannot store what happens during the millisecond in between.

A more important restriction regarding time is that it's limited. For our simulations they must at worst finish before the project is over and for our hardware based system, all calculations required to send another sample to the CODEC must be complete before it's time to send another sample. This later challenge is called the real-time requirement. Our CODECs sample at 48kHz and so there is 20.83 microseconds for the DSP to complete all necessary calculations for one sample. Obviously, this restricts whatever numerical method we use to solve the system.

3: Numerical Method Limitations - The most important choice we can make in designing our hardware-based communication system is what numerical method to use for solving the differential equations involved. We desire high-accuracy, but are realistically limited by timing constraints. This means that the numerical method should offer a high ratio of accuracy to processing requirements. More about numerical methods and the particular method we selected follows.

A.1 Numerical Methods to Solve Differential Equations

Using a discrete system to solve a continuous set of differential equations without a closed form solution creates the need for a numerical method. The fact that our continuous system in question is chaotic and therefore exhibits sensitive dependence on initial condition compounds the problem. In order to preserve the features of the continuous system in question, the numerically solved version must closely approximate the actual solution. With the effects of sensitive dependence on initial condition, we expect that the solution produced by a numerical method will not accurately follow the actual solution for a significant length of

time. However, we do expect and require that the general behavior of the system will be accurately modeled.

For example, suppose that we had the full solution for a continuous system and the differential equations that describe that solution. Given a numerical method that allows us to adjust the accuracy of the solution, we know that we can never set the accuracy high enough to match the real solution for a long time. Instead, we will be satisfied to set the accuracy high enough that general behavior is preserved. If solving the Lorenz system, we would want to see at least the attracting behavior and same pair of wings (amplitude, shape, etc.).

For all numerical methods, there is a trade off between system accuracy and processing complexity. However, not all methods use processing time as efficiently. For example, the most recognized numerical method is probably Euler's Method due to its simplicity (*not* its accuracy). Given an initial value problem,

$$\frac{dy}{dt} = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha. \tag{A.1}$$

Euler's Method attempts to solve for $y(b)$ by making a series of intermediate steps between a and b. The time between steps is called the *step size*. The method calculates the function value at the next time step based solely on the first derivative at the current step. Error can quickly accumulate and the numerical solution will no longer be close to the actual solution. See figure A.1 for what can happen to the solution along a simple curve.

Local truncation error measures the error at one step compared to the actual solution, assuming the value at the previous step was exactly correct. It depends on the particular system being solved, the step size, and also the particular step of the approximation (At some times the actual solution my be smooth or flat and at others it might have sharp curves indicating a quickly changing first derivative.). Reported as a measure of approximation efficiency, truncation error in Euler's Method is $O(h)$ [22]. Roughly speaking, this means that reducing the step size by a constant factor will also reduce error by the same factor. Also important to note is that for every time step, the derivative $f(t, y)$ must be calculated once. Numerical methods have been developed that are more efficient.

Out of many options, we use the fourth order Runge-Kutta algorithm at all times due to both its high approximation efficiency and high accuracy. For this algorithm, truncation error is $O(h^4)$ [22]. So, reducing the step size by a factor of two would improve error by a factor of 2^4 or sixteen. The fourth order Runge-Kutta algorithm evaluates the first derivative $f(t, y)$ four times per step to achieve this accuracy. Computation of the derivative is considered the most computationally expensive step for any numerical algorithm.

Suppose we want to improve the accuracy of Euler's method by a factor of 81. Then 81 steps must be taken and the derivative calculated 81 times. To achieve the same improvement with fourth order Runge-Kutta, three more steps are taken and the derivative is calculated only twelve times.

Another feature of the fourth order Runge-Kutta algorithm is that it's a one-step solver. This means that to calculate a step, only the state of the system at the immediately preceding step is needed. Some algorithms evaluate the next step based on a history of values at

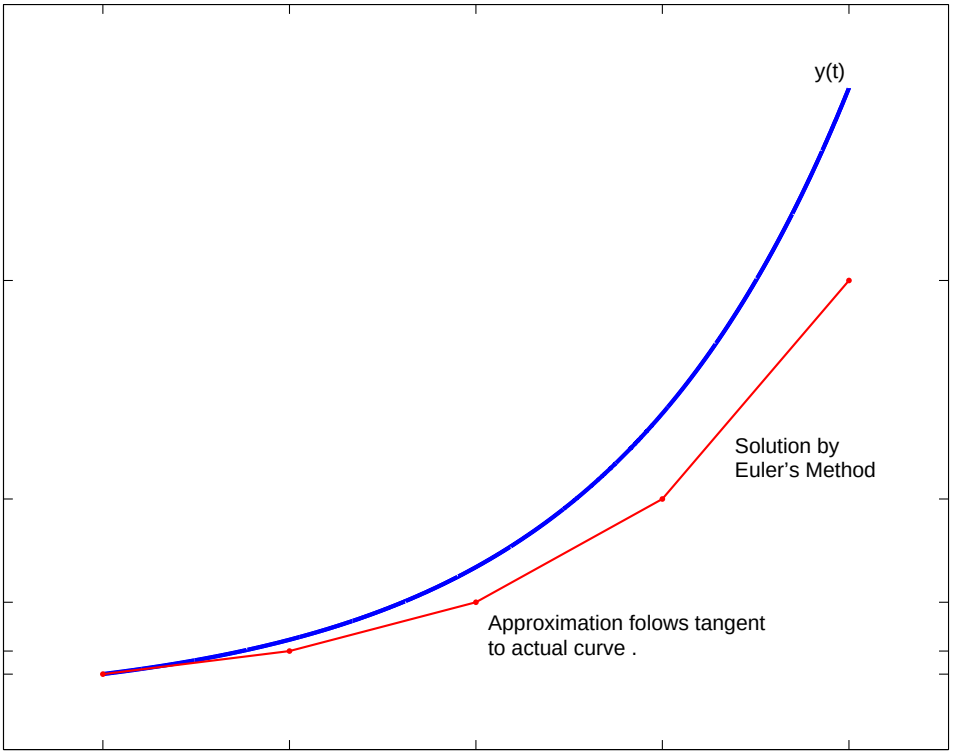

Figure A.1: Example solution using Euler's Method. Error is incurred at every time step and is not corrected for.

A.2 Fourth Order Runge-Kutta

To approximate the solution to the initial-value problem

$$y' = f(t, y), \quad a \le t \le b, \quad y(a) = \alpha,$$

select a positive integer N that will be the number of steps taken over the time interval.

Step 1 Set $h = (b - a)/N$, $t_0 = a$, *and* $w_0 = \alpha$.

Step 2 Set $i = 0$.

Step 3 Set

$$\begin{aligned} k_1 &= hf(t_i, w_i), \\ k_2 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1), \\ k_3 &= hf(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2), \\ k_4 &= hf(t_i + h, w_i + k_3), \\ w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \\ t_{i+1} &= a + (i + 1)h. \end{aligned}$$

Step 4 If $i = N - 1$, go to Step 6.

Step 5 Add 1 to i and go to Step 3.

Step 6 The procedure is complete, and w_i approximates $y(t_i)$, for each $i = 1, 2, \ldots, N$.

Appendix B

Using Compiled C Code in the MATLAB Environment

Several types of simulations for this project are computationally intensive. One problem we notice is that performing all of the necessary algorithms in the MATLAB environment is not fast enough. MATLAB is designed and optimized to perform operations on vectorized data. It is very good with this and comparable in performance to compiled C language code. MATLAB slows considerably when it is asked to perform conditional statements or looping. Looping is especially unavoidable in our simulations that performs the same complex operation on many different inputs. Its apparent that some simulations based solely on MATLAB code would not finish within the time constraints of the project.

We use an alternative method that offers substantially accelerated performance, but allows us to continue using the built-in MATLAB functions that make the environment so popular. (Visualization functions are just one example.) Several of our simulations use a method called mex-function programming. This method interfaces compiled C language code with the MATLAB environment. We find performance to be over 200 hundred times faster in compiled C than in MATLAB for some simulations. Jobs that currently run for about one day would obviously not be possible without a large cluster of computers to split the them up.

Our PCs that run the simulations consist of a 1.0 GHz AMD Athlon desktop with 512MB of RAM running Windows 98 and a Pentium III 500MHz laptop with 128MB of RAM running Windows 2000. We also ran several simulations on the eight processor Silicon Graphics time-sharing machine *superaero* run by CADIG in the Engineering Division.

B.1 A Brief Description of MEX Function Development

Converting an existing function written in MATLAB's own interpretive language to compiled C code takes five basic steps. Note that the Fortran language can be compiled for use with MATLAB as well. The Fortran language was not used in this project and will not be

discussed further.

1: The first step is to convert vectorized MATLAB code to explicit array code. One thing that must be kept in mind and can be frustrating is that the first value of a vector in MATLAB is indexed with a 1, but the first value in a C language array is indexed with a 0. Suppose we want to multiply three elements of a vector A by the scalar two and assign the result to a new vector B. In MATLAB this can be accomplished with the one-line command

```
B = 2*A;
```

To do the same thing in C language, the following lines might be used. Storage memory for B must first be allocated.

```
B[0] = 2*A[0];
B[1] = 2*A[1];
B[2] = 2*A[2];
```

The difference in the complexity of the two statements would be even more substantial if we did not know the length of the vector A. The MATLAB line would not change, but the C code would first have to determine the size of the vector and then perform a looping assignment once for each unit in the vector.

2: The next step is to utilize the mex-file templates provided with the MATLAB distribution. These templates include the C code that is common to all mex-files. Most importantly, they establish a link between the compiled mex file and memory space in the MATLAB environment. This way variables that exist in MATLAB can be passed to the mex-file and visa versa.

3: Adapt the basic mex template to the specific type and number of arguments passed to and returned from the mex function. In MATLAB, we do not have to worry about accidentally trying to assign a character to an array that is supposed to store numbers. The environment will catch the error and let us know. Similarly, MATLAB will not let us read a value that is not in the array. For the four element array `B`, calling `B(25)` will generate an error but in C, calling `B[24]` will compile without error and will give a result. Unfortunately, it will be unpredictable and not what we might have intend. In compiled C, there are just not as many safeguards. A significant part of most mex functions is simply ensuring that the arguments passed to the function and returned from the function are of the type and size expected.

4: Once the mex file has been set up to accept the desired parameters of the appropriate type and return the correct results, we insert our own C algorithms. Often this is a surprisingly small fraction of the code. If the mex file is to be more efficient than the equivalent algorithm in MATLAB's interpretive language, then the time saved by the compiled algorithm must outweigh the added overhead for argument checking. It almost always is.

5: The last step before the mex file can be called from MATLAB is to compile it. MATLAB's command `mex` does this. The first time the command is run, it gives

the user the option to use MATLAB's own `lcc` compiler or another compiler installed on the computer such as Microsoft Visual C++. We use the Microsoft compiler. As long as everything compiles without error, the result is a dynamically linked library (.dll) file that can be called from the MATLAB environment as though it were a regular function. For example, a simple function that returns the root mean square value of a series of numbers might be written in the C language file calcVrms.c. Once, compiled, the file calcVrms.dll can be called in MATLAB with the command `Vrms = calcVrms([2, 3, 4 -1, 2, 0, 2])`.

B.2 A Simple Mex Function

Below is a simple mex function that computes the RMS value of an input array. Comments describe the main components.

```
/*=================================================================
 *
 * calcVrms.c Calculates the root mean square value of an input array
 *
 * The calling syntax is:
 *
 * Vrms = calcVrms( vin )
 * Vrms - output RMS value, 1 x 1.
 * vin  - input array, ususually a series of voltage values, 1 x N.
 *
 * This is a MEX-file for MATLAB.
  *=================================================================*/

#include "mex.h"   /*A library file provided by MATLAB that declares many
 of the functions for checking input arguments and
 establishing memory for output arguments below. */
#include <math.h>  /*A standard C library that gives access to the sqrt()
 function used in the root mean square algorithm.*/

/* Input Arguments */
#define V_IN prhs[0]  //symbol assigned to the memory location of the
 //input argument.

/* Output Arguments */
#define VRMS_OUT plhs[0]  //symbol assigned to the memory location of
 //the output argument.
```

```
void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )
{
    double *vinptr, *vrmsptr;  //declare any variables necessary,
   //including pointers to access.
    double sum;    //input and output arguments.
unsigned int m, N, i;

/* Check for proper number of arguments */
    if (nrhs != 1) {
mexErrMsgTxt("One input argument requrired.");
//Ensure that proper number of input arguments
//passed or send an error back to the MATLAB environment.
    } else if (nlhs > 1) {
mexErrMsgTxt("Too many output arguments.");
//Often, the number of output argumetns is optional,
    } //but there is always a minimum number.

    /* Check the dimensions and type of input arguments */

    /* V_IN must be 1xN*/
m = mxGetM(V_IN); //get number of rows
    N = mxGetN(V_IN); //get number of columns
    if (!mxIsDouble(V_IN) || mxIsComplex(V_IN) || m!= 1) {
mexErrMsgTxt("calcVrms() requires that vin be a 1xN vector.");
/*The conditional statement checks to make sure that the input is of
  type double and is only real values.  MATLAB stores real and comlex
  values in two different formats.  The input argument must have one
  row, but the number of columns is up to the user.*/
    }

    /* Assign pointers to the various input parameters */
    vinptr   = mxGetPr(V_IN);  //this is the pointer used to access the
   //array in the code

    /* Create a matrix for the return arguments */
    VRMS_OUT = mxCreateDoubleMatrix( 1, 1, mxREAL); /*memory is alocated
  to store the output argument*/
    vrmsptr  = mxGetPr(VRMS_OUT); /*this pointer is used to access the
  allocated memory and pass the return
  argument back to the MATLAB environment.
  Size of the output array depends on the
```

```
  size of the input argumetn (N). */

/* and finally, the Vrms algorithm */
/* ------------------------------- */
sum = 0;
for( i = 0; i < N; i++)  //this loop calculates the sum of the square
     //of all values in the length-N array.
{
sum += vinptr[i] * vinptr[i];
}

vrmsptr[0] = sqrt( sum / N );  //compute square root of the average.

    return;
}
```

Appendix C

Modulating for Propagation

For wireless and many wire-based systems, a baseband signal must be modulated to a higher frequency. There are several reasons for doing this. One is so the signal will propagate through the channel and another is so the transmission will fit in an allotted bandwidth.

There are several methods for doing this. Some methods are advantageous because it's simple to construct a modulator and demodulator to shift the signal spectrum up to the desired frequency and back down again. Other methods require a more complex modulator and demodulator but offer other advantages like using half-the bandwidth.

Sending one-half of the baseband bandwidth is called single sideband (SSB). We partially develop the mathematical framework of SSB by the Hilbert transform and analytic signal method. Figure C.1 illustrates the difference between single sideband and basic modulation techniques.

C.1 The Hilbert Transform and the Analytic Signal

First, some notation: Let $m(t)$ be the real-valued message that we desire to transmit. The Hilbert transform of the message is denoted $\hat{m}(t)$.
Let $m_+(t)$ denote the analytic signal version of $m(t)$. The analytic signal $m_+(t)$ has no power at negative frequencies. They are suppressed in the frequency domain.

The Hilbert transform is defined as the following convolution,

$$\hat{m}(t) = m(t) * h(t), \quad \text{where } h(t) = \frac{1}{\pi t}. \tag{C.1}$$

The Fourier transform of $h(t)$ is denoted $H(f)$, or $h(t) \overset{\mathcal{F.T.}}{\Longleftrightarrow} H(f)$. It is defined by

$$H(f) = \begin{cases} j, & f \leq 0 \\ 0, & f = 0 \\ -j, & f \geq 0. \end{cases} \tag{C.2}$$

The analytic version of a real valued signal can be realized by summing the original signal with j, $(\sqrt{-1})$, times its Hilbert transform.

$$m_+(t) = m(t) + j\hat{m}(t). \tag{C.3}$$

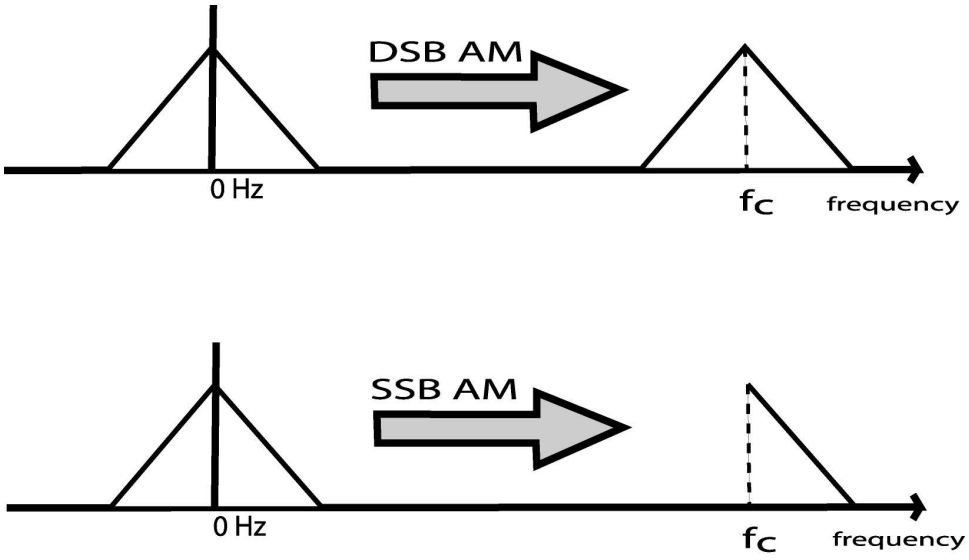

Figure C.1: The top diagram illustrates the shifting of both sides of a signal's frequency spectrum in double-sideband amplitude modulation (DSB AM). The bottom diagram shows the shifting for single-sideband amplitude modulation (SSB AM); in this case the upper sideband is reallized (USSB). SSB uses one-half the bandwidth as DSB to send the same message. The original signal can be recovered from either method, but the receiver for SSB is more complex.

$$m\ t \qquad m\ t \qquad m\ t\ * \frac{1}{\pi t} \qquad m\ t \qquad m\ t \overset{\mathcal{F}.\mathcal{T}.}{\Longleftrightarrow} M\ f$$

$$m\ t \quad \overset{\mathcal{F}.\mathcal{T}.}{\Longleftrightarrow} \quad M\ f \qquad M\ f\ H\ f\ .$$

$$M\ f \quad \begin{cases} jM\ f\ , & f \leq \\ \quad , & f \\ -jM\ f\ , & f \geq \end{cases} .$$

$$m_+\ t \qquad m\ t \qquad jm\ t \overset{\mathcal{F}.\mathcal{T}.}{\Longleftrightarrow} M_+\ f \qquad M\ f \qquad jM\ f\ .$$

$$
\begin{aligned}
M_+(f) &= \begin{cases} M(f) + j^2 M(f), & f \leq 0 \\ M(f) + j(0), & f = 0 \\ M(f) - j^2 M(f), & f \geq 0. \end{cases} \\
&= \begin{cases} 0, & f \leq 0 \\ M(f), & f = 0 \\ 2M(f), & f \geq 0. \end{cases}
\end{aligned} \tag{C.7}
$$

Energy is eliminated from the negative frequencies of the signal.

C.2 Single Sideband Modulation

Standard amplitude modulation is achieved by multiplying a message, $m(t)$, by a sinusoidal carrier $cos(\omega_c t)$. For single sideband modulation, either the positive or negative frequencies must be stripped from both the message and the modulating sinusoid. We will develop the upper single sideband (USSB) technique where energy below the carrier frequency is suppressed, but the method for lower single sideband (LSSB) is similar.

Let $\gamma(t) = \cos(\omega_c t)$ be the modulating sinusoid, where ω_c is the carrier frequency. The analytic version of this function is,

$$\gamma_+(t) = \gamma(t) + j\hat{\gamma}(t). \tag{C.8}$$

The Hilbert transform of a cosine is known. It's simply a 90° phase shift,

$$\cos(\omega_c t) \overset{\mathcal{H}.\mathcal{T}.}{\Longleftrightarrow} \sin(\omega_c t), \tag{C.9}$$

so $\hat{\gamma}(t) = \sin(\omega_c t)$. Then the analytic signal is,

$$\gamma_+(t) = \cos(\omega_c t) + j \sin(\omega_c t),$$

which is Euler's Identity,

$$\gamma_+(t) = e^{j\omega_c t}. \tag{C.10}$$

The USSB signal $g(t)$ is the product of the analytic message $m_+(t)$ and the analytic carrier $\gamma_+(t)$,

$$
\begin{aligned}
g(t) &= m_+(t)\gamma_+(t) \\
&= m_+(t)e^{j\omega_c t} \\
&= m_+(t)[\cos(\omega_c t) + j \sin(\omega_c t)].
\end{aligned}
$$

It is useful to expand this expression into its real and imaginary parts.

$$
\begin{aligned}
g(t) &= [m(t) + j\hat{m}(t)][\cos(\omega_c t) + j \sin(\omega_c t)] \\
&= m(t)\cos(\omega_c t) + jm(t)\sin(\omega_c t) + j\hat{m}(t)\cos(\omega_c t) - \hat{m}(t)\sin(\omega_c t).
\end{aligned} \tag{C.11}
$$

Only the real part of this signal is transmitted. We denote the real part of $g(t)$ by $\Psi(t)$,

$$\Psi(t) = \mathcal{R}e\{g(t)\} = m(t)\cos(\omega_c t) - \hat{m}(t)\sin(\omega_c t), \tag{C.12}$$

Expressed differently,

$$\Psi(t) = m(t)\gamma(t) - \hat{m}(t)\hat{\gamma}(t), \quad \text{where} \quad \begin{array}{rcl} \gamma(t) & = & \cos(\omega_c t) \\ \hat{\gamma}(t) & = & \sin(\omega_c t). \end{array} \tag{C.13}$$

We need Bedrosian's Theorem relating the Hilbert transform of two signals with non-overlapping spectra for the work to following.

Theorem C.2.1. ***Bedrosian's Theorem****[23] Given, the non-overlapping spectra signals $y(t)$ and $z(t)$, then there is some frequency W such that the spectrum of $y(t)$ vanishes for $|f| > W$ and the spectrum of $z(t)$ vanishes for $|f| < W$.*

$$\begin{array}{rcl} Y(f) & = & 0 \quad for \quad |f| > W \\ Z(f) & = & 0 \quad for \quad |f| < W \end{array}$$

In the product $y(t)z(t)$, $y(t)$ is the low-pass term and $z(t)$ is the high-pass term.

Bedrosian's Theorem shows the following property of the Hilbert transform (denoted $\mathcal{H}\{\}$) of the product of two non-overlapping spectra signals:

$$\mathcal{H}\{y(t)z(t)\} = y(t)\mathcal{H}\{z(t)\}. \tag{C.14}$$

The process of recovering a USSB signal utilizing Bedrosian's theorem is partially shown in [24] and reported here.

We expect to have a band limited baseband signal $m(t)$ and will choose the carrier signal $\gamma(t)$ such that Bedrosian's theorem can be applied. (The lowest frequency component of $\gamma(t)$ is higher than the highest component of $m(t)$.) Then,

$$\begin{array}{rcl} \hat{\Psi}(t) & = & m(t)\hat{\gamma}(t) - \hat{m}(t)\hat{\hat{\gamma}}(t) \\ & = & m(t)\hat{\gamma}(t) + \hat{m}(t)\gamma(t), \end{array}$$

and,

$$[\hat{\Psi}^2(t)] = m^2(t)\hat{\gamma}^2(t) + 2\hat{m}(t)\gamma(t)m(t)\hat{\gamma}(t) + \hat{m}^2(t)\gamma^2(t),$$

$$[\Psi^2(t)] = m^2(t)\gamma^2(t) - 2\hat{m}(t)\gamma(t)m(t)\hat{\gamma}(t) + \hat{m}^2(t)\hat{\gamma}^2(t),$$

and through a lot of simplification,

$$[\Psi^2(t) + \hat{\Psi}^2(t)] = m^2(t) + \hat{m}^2(t).$$

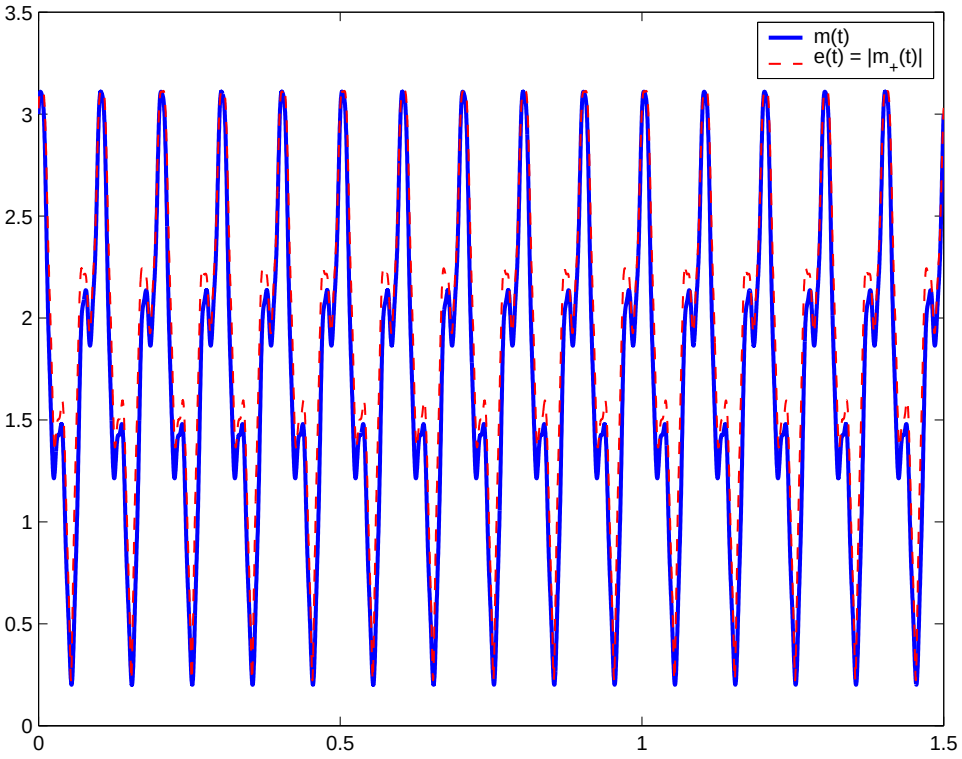

Figure C.2: Original message $m(t)$ is a sum of three sinusoids and a DC offset to ensure $m(t) > 0$. The dashed line is the recovered signal from the complex envelope $|m_+(t)|$.

$$e\ t \qquad {}^{2}\ t \qquad {}^{2}\ t \quad {}^{1/2}$$

$$m^2\ t \qquad m^2\ t \quad {}^{1/2}$$

$$|m_+\ t\ |. \qquad 1$$

$$m\ t \qquad |m_+\ t\ | \qquad m\ t\ >$$

Appendix D

Project Code

Here we include some of the more significant algorithms and programs developed to implement our system in hardware and perform the parameter space search framework discussed.

D.1 Basic DSP Drive System

The following C language code is compiled for the TI 'C6711 digital signal processor. It serves as a basic drive system for the response system described in section D.2.

D.1.1 Code for Digital to Analog Converter

The digital to analog converter and analog to digital converter (CODEC collectively) are interfaced to the 'C6711 DSP through the on chip serial port McBSP0. The drive system uses only the analog to digital converter. Samples are converted at the sampling rate of 48,000 samples per second. To meet real time requirements, the DSP program code must write a new sample to the CODEC through the serial port at the sampling rate before each new sample is needed.

```
/*================================================================
 *
 * driveCODEC.c
 *
 *      codec support routines for C6X11 dsk w/ PCM3006 daughterboard
 *      based on code developed by Michael G. Morrow, 2001.
 *      Our additions handle sending Lorenz function samples to the CODEC.
 *
 * The calling syntax is:
 *      function McBSP1_Tx_ISR() is initiated by an interupt generated by
 * the serial port when it is ready for another sample.
 *================================================================*/
```

```
#include <c6x.h>
#include "c6211dsk.h"
#include "codec.h"

#define LEFT  0
#define RIGHT 1

// Data is received from the PCM3006 codec as 2 16-bit words (left/right)
// packed into one 32-bit word.  The union allows the data to be accessed
// as a single entitiy when transferring to and from the serial port, but
// still be able to manipulate the left and right channels independently.

volatile union {
unsigned int uint;
short channel[2];
} TxData;

interrupt void McBSP1_Tx_ISR()
{
McBSP *port  = McBSP1_Base;   //Port address of serial port
float * lorenzOutput;
float x, y;

lorenzOutput = rk45step(); /*get next Lorenz system state for output*/

x = lorenzOutput[0];
y = lorenzOutput[1];
x = x * 32767.0;   //scale to floating point numbers (-1 to +1)
y = y * 32767.0;   //to short range.

TxData.channel[LEFT] = (short)x;  //assign both stereo channels
TxData.channel[RIGHT] = (short)y; //to one 32 bit data unit

port->dxr = TxData.uint;// send stereo sample to CODEC via serial port
}
```

D.1.2 RK-45 Algorithm

```
/*================================================================
 *
 * rk47step.c iterates one step of the Lorenz system using
```

```
 * the Runge-Kutta 4-5 numerical method.
 *
 * The calling syntax is:
 *
 * newState[3] = rk45step();
 * newState[3] - new [x,y,z] state after one time step by RK 4-5.
 *      note: state[3] is a static variable that holds the sytem's current
 *       state between calls of this function.
 *=================================================================*/

static float state[3];
float STEPSIZE = 5e-2;  //time step of numerical methods (fake seconds)

const float* rk45step()
{

float input[3], k1[3], k2[3], k3[3], k4[3]; /*factors
of Runge-Kutta method*/
    /*k1 term*/
    input[0] = state[0];
    input[1] = state[1];
    input[2] = state[2];
    lorenz(input, k1);
    k1[0] = k1[0]*STEPSIZE;
    k1[1] = k1[1]*STEPSIZE;
    k1[2] = k1[2]*STEPSIZE;

    /*k2 term*/
    input[0] = state[0] + k1[0] / 2.0;
    input[1] = state[1] + k1[1] / 2.0;
    input[2] = state[2] + k1[2] / 2.0;
    lorenz(input, k2);
    k2[0] = k2[0]*STEPSIZE;
    k2[1] = k2[1]*STEPSIZE;
    k2[2] = k2[2]*STEPSIZE;

    /*k3 term*/
    input[0] = state[0] + k2[0] / 2.0;
    input[1] = state[1] + k2[1] / 2.0;
    input[2] = state[2] + k2[2] / 2.0;
    lorenz(input, k3);
    k3[0] = k3[0]*STEPSIZE;
    k3[1] = k3[1]*STEPSIZE;
```

```
    k3[2] = k3[2]*STEPSIZE;

    /*k4 term*/
    input[0] = state[0] + k3[0];
    input[1] = state[1] + k3[1];
    input[2] = state[2] + k3[2];
    lorenz(input, k4);
    k4[0] = k4[0]*STEPSIZE;
    k4[1] = k4[1]*STEPSIZE;
    k4[2] = k4[2]*STEPSIZE;

    /*update state*/
    state[0] = state[0] + ( k1[0] +2.0*k2[0] + 2.0*k3[0] + k4[0] )/6.0;
    state[1] = state[1] + ( k1[1] +2.0*k2[1] + 2.0*k3[1] + k4[1] )/6.0;
    state[2] = state[2] + ( k1[2] +2.0*k2[2] + 2.0*k3[2] + k4[2] )/6.0;

return state;
}
```

D.1.3 Lorenz System

```
/*================================================================
 *
 * driveLorenz.c   evaluates differential ODEs describing the Lorenz system
 *   using constant parameter values.
 *
 * The calling syntax is:
 *
 * driveLorenz( y[3], yprime[3] )
 * y[3] - a three element float array holding [x,y,z] state values.
 * yprime[3] - a three element float array that will be used to store
 * the evaulated derivative values [dx,dy,dz].
 *================================================================*/

volatile float SIGMA = 16.0;
volatile float RHO = 45.6;
volatile float BETA = 4.0;

volatile float A = 40;   /*scale factor u = x/A such that
   system is in -1/+1V range */
```

```
void lorenz( const float * y, float * vprime)
{

/*The ODEs of the Lorenz system in C language notation.
y[0] = x; y[1] = y; y[2] = z; */

vprime[0] = SIGMA * ( y[1]-y[0] );
vprime[1] = (RHO * y[0] - y[1] - A*y[0]*y[2]);
vprime[2] = (A*y[0]*y[1] - BETA * y[2]);

return;
}
```

D.2 Basic DSP Response System

The following C language code is compiled for the TI 'C6711 digital signal processor. It serves as a basic response system to achieve synchronization with a drive system described in section D.1.

D.2.1 Code for CODEC

For the receiver to meet real time requirements, the DSP program code must receive a new sample from the drive system, process the next time step of the influenced Lorenz system using the Runge-Kutta 4-5 method, and then output the updated state.

```
/*=================================================================
 *
 * responseCODEC.c
 *
 *      codec support routines for C6X11 dsk w/ PCM3006 daughterboard
 *      based on code developed by Michael G. Morrow, 2001.
 *      Our additions handle responding to influence from the drive
 * system and sending receiver sponse to the output for comparison.
 *
 * The calling syntax is:
 *      functions McBSP1_Tx_ISR() and McBSP1_Rx_ISR() are initiated by
 * interupts generated by the serial port when it is ready for
 * another sample.
 *=================================================================*/

#define LEFT  0
```

```
#define RIGHT 1
#define SHORT_FLOAT_SCALE 32767.0
// Data is received from the PCM3006 codec as 2 16-bit words (left/right)
// packed into one 32-bit word.  The union allows the data to be accessed
// as a single entitiy when transferring to and from the serial port, but
// still be able to manipulate the left and right channels independently.

volatile union {
unsigned int uint;
short channel[2];
} TxData;

volatile union {
unsigned int uint;
short channel[2];
} RxData;

static float influence; /*term received from transmitter */
static short carrythrough;  /*one channel of output will simply pass
  through the input channel.*/

interrupt void McBSP1_Rx_ISR()
{
/*sample influence signal originating at transmitter*/
McBSP *port  = McBSP1_Base;
RxData.uint = port->drr; /* get input data from serial port */
carrythrough = RxData.channel[RIGHT];//get input data for passthrough
influence = (float)RxData.channel[RIGHT]/SHORT_FLOAT_SCALE;
   /* scale input to -1 to +1V */

/*Ensure that receive and transmit interupts alternate in case
  the two interupt service routines would otherwise be called
  asynchronously.*/
IER ^= 0x1800;  /*exclusive OR operation turns Rx_ISR off
  and Tx_ISR on so that the next ISR will be a
  transmit*/
}

interrupt void McBSP1_Tx_ISR()
{
short codecOut;
float xout;
McBSP *port  = McBSP1_Base;
```

```
/*output coupled system */
rk45step(&influence, &xout);//get next state based on influence
codecOut = xout * SHORT_FLOAT_SCALE;//rescale new state to short rng
TxData.channel[LEFT] = codecOut;  /*send new state based on influence
to CODEC */
TxData.channel[RIGHT] = carrythrough; //passthrough for comparison
    port->dxr = TxData.uint; // send stereo channels to serial port
    /*set next interupt to be Rx*/
IER ^= 0x1800;  /*exclusive OR operation turns Rx_ISR on
  and Tx_ISR off so that the next ISR will be a
  receive*/
}
```

D.2.2 Lorenz System with Coupling

```
/*================================================================
 *
 * responseLorenz.c evaluates differential ODEs describing the
coupled Lorenz system using constant parameter values.
 *
 * The calling syntax is:
 *
 *    coupledLorenz( y[3], x_infl*, yprime[3] )
 *    y[3] - a three element float array holding [x,y,z] state values.
 *    x_infl* - a pointer to the influence term.
 *    yprime[3] - a three element float array that will be used to store
 * the evaulated derivative values [dx,dy,dz].
 */

const float SIGMA = 16.0;
const float RHO = 45.6;
const float BETA = 4.0;

volatile float A = 40;   /*scale factor u = x/A such that
   system is in -1/+1V range */

void coupled_lorenz(const float* y, const float* x_coupled, float* vprime)
{
/*The ODEs of the Lorenz system in C language notation.
y[0] = x; y[1] = y; y[2] = z;
x_coupled is the value received from the drive system*/
```

```
vprime[0] = SIGMA * ( y[1]-y[0] );
vprime[1] = (RHO * *x_coupled - y[1] - A* *x_coupled*y[2]);
vprime[2] = (A * *x_coupled *y[1] - BETA * y[2]);
return;
}
```

D.3 Code for MATLAB simulations

D.3.1 A Generic Lorenz System Function

We used several different variants of the following function. This one allows the Lorenz system to be iterated for as long as the user desires, using a specified step size. The function accepts as an input a parameter set for the Lorenz system and returns all three state variables instead of just the transmitted x. This is useful for producing phase portraits but, in general, unnecessarily uses memory and processor time for most simulations.

```
/*=================================================================
 *
 * iterate3DLorenz.c   Iterates Lorenz system using
 *    4th-Order Runge-Kutta  method
 *
 * The calling syntax is:
 *
 * [v Vf] = iterateLorenz(v0, parameters, STEPSIZE, sampbit )
 * v - output state vector.  sampbit x 3.
 * Vf - final system state at end of bit. 1 x 3.
 * v0 - initial state vector 1 x 3
 * parameters - [sigma, rho, beta] for receiver 1x3.
 * STEPSIZE - resolution for RK45 method
 * sampbit - samples per bit
 *
 * This is a MEX-file for MATLAB.
 * Copyright 2001, Noah F. Reddell
 *
 *=================================================================*/
/* $Revision: 2.2 $ */

#include "mex.h"

void  lorenz(const double *, double *);  //function included in this file
```

```
/****Input parameter checking removed for space considerations ****/

void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )
{
    double *v0ptr, *vptr, *vfptr, *parameters;
    double STEPSIZE, sampbit;
    unsigned int m,n, i, N;
double input[3], k1[3], k2[3], k3[3], k4[3]; /*R-K factors*/
    double x0, y0, z0;
//
//...
//

/* Create a matrix for the return arguments */
    V_OUT = mxCreateDoubleMatrix( (int)sampbit, 3, mxREAL);
    vptr = mxGetPr(V_OUT); /*output pointer, sampbit x 3 array*/

VF_OUT = mxCreateDoubleMatrix( 1, 3, mxREAL);
    vfptr = mxGetPr(VF_OUT); /*output pointer, 1 x 3*/

/*assign initial values to output array*/
x0 = v0ptr[0];  /*set initial state */
y0 = v0ptr[1];
z0 = v0ptr[2];

/*set lorenz system parameters*/
RHO = parameters[1];
BETA = parameters[2];
SIGMA = parameters[0];

//take N time steps using RK-45 method
for(i= 0; i < N; i++)
{
/*k1 term*/
input[0] = x0;
input[1] = y0;
input[2] = z0;
lorenz(input, k1);
k1[0] = k1[0]*STEPSIZE;
k1[1] = k1[1]*STEPSIZE;
k1[2] = k1[2]*STEPSIZE;
```

```
/*k2 term*/
input[0] = x0 + k1[0] / 2.0;
input[1] = y0 + k1[1] / 2.0;
input[2] = z0 + k1[2] / 2.0;
lorenz(input, k2);
k2[0] = k2[0]*STEPSIZE;
k2[1] = k2[1]*STEPSIZE;
k2[2] = k2[2]*STEPSIZE;

/*k3 term*/
input[0] = x0 + k2[0] / 2.0;
input[1] = y0 + k2[1] / 2.0;
input[2] = z0 + k2[2] / 2.0;
lorenz(input, k3);
k3[0] = k3[0]*STEPSIZE;
k3[1] = k3[1]*STEPSIZE;
k3[2] = k3[2]*STEPSIZE;

/*k4 term*/
input[0] = x0 + k3[0];
input[1] = y0 + k3[1];
input[2] = z0 + k3[2];
lorenz(input, k4);
k4[0] = k4[0]*STEPSIZE;
k4[1] = k4[1]*STEPSIZE;
k4[2] = k4[2]*STEPSIZE;

/*update state*/
x0 = x0 + ( k1[0] +2.0*k2[0] + 2.0*k3[0] + k4[0] )/6.0;
y0 = y0 + ( k1[1] +2.0*k2[1] + 2.0*k3[1] + k4[1] )/6.0;
z0 = z0 + ( k1[2] +2.0*k2[2] + 2.0*k3[2] + k4[2] )/6.0;

/*record the transmitted output */
vptr[i] = x0;
vptr[N+i] = y0;
vptr[2*N+i] = z0;
}

/* update final state at end of bit */
vfptr[0] = x0;
vfptr[1] = y0;
vfptr[2] = z0;
```

```
    return;
 }

/*lorenz system*/

void lorenz( const double * y, double * vprime)
{

/*The ODEs of the Lorenz system in C language notation.
y[0] = x; y[1] = y; y[2] = z; */
vprime[0] = SIGMA * ( y[1] - y[0] );
vprime[1] = (RHO*y[0] - y[1] - y[0]*y[2]);
vprime[2] = (y[0]*y[1] - BETA*y[2]);
return;
}
```

D.3.2 Lyapunov Exponent by Variational Equation Method

Code excerpts from the MEX function we use to estimate the Lyapunov exponent of our system for various parameter sets.

```
/*=================================================================
 *
 * lorenzLyapunovVM.c : calcualates dominant lyapunov exponent of
 *   Lorenz equations for given parameters via
 * the variational equations
 *
 * The calling syntax is:
 *
 * [h xRMS] = lorenzLyapunovVM(parameters, V0, numAvg, steps)
 * Output:  h - calculated dominate lyapunov exponent (average)
 * xRMS - RMS value of the x state over the integration
 *
 * Inputs:  parameters - [sigma, rho, beta] for system 1x3.
 * v0 - initial state vector 1 x 3
 * numAvg - number of averages in calculating exponent.
 * steps - number of iterations to make for one time unit.
 * (also 1/stepsize of numerical solver)
 *
 * This is a MEX-file for MATLAB.
 * Copyright 2001, Noah F. Reddell
```

```
  *=================================================================*/
/* $Revision: 2.5 $ */

void  lorenzAndVar(const double *, double *);  /* ( Y , Yprime)*/
void  time1LorenzandVarMap(const double *, double *, const double *,
 ...     double *, double *); /* ( V0, Vf, D0, Df, &xSquaredSum ) */

/*to be used globally*/
double SIGMA, RHO, BETA, STEPSIZE;
unsigned int Nsampbits;
double xSquaredSum = 0;  //used to calculate average x-value

void mexFunction( int nlhs, mxArray *plhs[],
  int nrhs, const mxArray*prhs[] )
{
    double *V0, *parameters, *hptr, *lambda_m, *xRMSPtr;
    double sampbit, numAvg, Df_length;
    unsigned int m,n, M;
double D0[3], Vf[3], Df[3], V0internal[3];
mxArray *L_ARRAY;

//
// ... argument conditioning removed for space considerations
//
/* Create a matrices for the return arguments */
    H_OUT = mxCreateDoubleMatrix( 1, 1, mxREAL);
    hptr =  mxGetPr(H_OUT); /*output pointer, 1 x 1*/

XRMS_OUT = mxCreateDoubleMatrix( 1, 1, mxREAL);
    xRMSPtr =  mxGetPr(XRMS_OUT); /*output pointer, 1 x 1*/

/*create storage for individual growth factors */
L_ARRAY = mxCreateDoubleMatrix( M, 1, mxREAL);
    lambda_m =  mxGetPr(L_ARRAY); /*output pointer, M x 1*/

/*set lorenz system parameters*/
SIGMA = parameters[0];
RHO   = parameters[1];
BETA  = parameters[2];

#define D0_Length 1.73205080756888 //sqrt(3) leads to unit length D0
D0[0] = D0_Length; D0[1] = D0_Length; D0[2] = D0_Length;
/*initial separation delta vector*/
```

```
V0internal[0] = V0[0]; V0internal[1] = V0[1]; V0internal[2] = V0[2];

xSquaredSum = 0;

for(m = 0; m < M; m++)  //call time-1 map for number of averages
{
time1LorenzandVarMap(V0internal, Vf, D0, Df, &xSquaredSum);
/*calculate vector length*/
Df_length = sqrt(Df[0]*Df[0] + Df[1]*Df[1] + Df[2]*Df[2]);
lambda_m[m] = Df_length ; /*record growth for this time-1 step*/

D0[0] = (Df[0]/Df_length); D0[1] = (Df[1]/Df_length);
  D0[2] = (Df[2]/Df_length);/*rescale to unit length */
V0internal[0] = Vf[0]; V0internal[1] = Vf[1];
  V0internal[2] = Vf[2];  /*continue iteration where left off */
}

for(m = 0; m < M; m++)
hptr[0] += log(lambda_m[m]);
hptr[0] = hptr[0] / M;   /*return average logarithmic growth
after M iterations */
    xRMSPtr[0] = sqrt(xSquaredSum / (Nsampbits * M));//root mean square
 //for system over avg
return;
 }

 /*time-1 map of Lorenz System and Variational Equations*/
void  time1LorenzandVarMap(const double * V0, double * Vf,
const double * D0, double *Df, double * xSquaredSumPtr)
/* ( V0, Vf, D0, Df, xSum* ) */
{
double x0, y0, z0, input[6], k1[6], k2[6], k3[6], k4[6];//RK factors
double deltax0, deltay0, deltaz0;
unsigned int i;

/*assign initial values to output array*/
x0 = V0[0];  /*set initial state of lorenz system*/
y0 = V0[1];
z0 = V0[2];

deltax0 = D0[0];  /*set initial state of variational equations */
deltay0 = D0[1];
```

```
deltaz0 = D0[2];


for(i= 0; i < Nsampbits; i++)
{
/*RK-45 to develop six dimensional Lorenz and variational system*/

//add to variable used in calculating RMS of system.
xSquaredSumPtr[0] += x0*x0;
}
/* update final state at end of bit */
return;
 }

/*lorenz system with variational parts added.*/
void lorenzAndVar( const double * y, double * vprime)
{
/* input: state of system y -...
output: derivative evalutate at state y
*/

/*The system of differtial equations and corresponding variational eqns
ddeltax/dt =
ddeltay/dt =
ddeltaz/dt =
*/

/*Lorenz System*/
vprime[0] = SIGMA * ( y[1]-y[0] );
vprime[1] = (RHO * y[0] - y[1] - y[0]*y[2]);
vprime[2] = (y[0]*y[1] - BETA * y[2]);

/*Variational Equations*/
vprime[3] = SIGMA * ( -y[3] + y[4] );
vprime[4] = (RHO * y[3] - y[5]*y[0] - y[3]*y[2] - y[4]);
vprime[5] = (y[4]*y[0] + y[3]*y[1] - BETA * y[5]);

return;
}
```

D.3.3 Bit Energy Search

The following MATLAB script excerpt is used to calculate the average bit energy associated with one fixed parameter set and a variable parameter set chosen from candidates with a positive Lyapunov exponent.

```
%-----------------------------------------------------------------
%uniformSearchX.m
%find good set of parameters for use with our system.  Test only those
%parameter sets that yield a positive lyapunov exponent.

%establish paramset as a comparison
paramsetA = [16.0; 45.6; 4.0];
AScale = 1.263725811751014e+001;%Vrms value system with this parameter set

SPB = 100;  %number of iterations between bits (samples per bit)
%set initial values

A = 500; %number of bits to send for a good average.
BitEnergyResults = zeros(S,R,B); %will store results of search
bit_energies = zeros(1,A);
V0 = [1;1;1];  VF = V0;
STEPSIZE = 0.01;

%below, H(s,r,b) is the results vector of our Lyapunov exponent search

for s=1:S
    for r=1:R
        for b=1:B
            if( (H(s,r,b)) > 0.5 & (Vrms(s,r,b)) ) %only interested in
%Vrms if exponent is positive( 0.5 for confidence )
                paramsetB = [SIGMA(s); RHO(r); BETA(b)];
                message = round(rand(1,A)); %generate random binary bit
                BScale = Vrms(s,r,b); %scale systems for same avg power
                VF = V0;  %wrap-around initial condition
                %-------------------------------------------------------
     for a = 1:A
       if(message(a))  %one-bit transmit
         vb = LorenzDriveSystem(VF, paramsetB, BScale, STEPSIZE, SPB);
         [va VF] = LorenzDriveSystem(VF, paramsetA, AScale, STEPSIZE, SPB);
         bit_energies(a) = sum((va-vb).^2); %sum of squares
       else            %zero-bit transmit
```

```
          va = LorenzDriveSystem(VF, paramsetA, AScale, STEPSIZE, SPB);
          [vb VF] = LorenzDriveSystem(VF, paramsetB, BScale, STEPSIZE, SPB);
          bit_energies(a) = sum((va-vb).^2); %sum of squares
        end
      end
          BitEnergyResults(s,r,b) = mean(bit_energies);
                 %-------------------------------------------------------
             end
        end
    end
    disp(s)
end
```

D.3.4 Bit Error Probability Simulation

The following MATLAB script excerpt is used to calculate the bit error probability for our system under the influence of a range of Gaussian noise power levels.

```
%-----------------------------------------------------------------------
%runBER.m
%test bit error probability of Lorenz-based chaotic communication system
%for various noise power levels.

STEPSIZE = 0.01;

%AScale = 0.3160;
%BScale = 0.0565;

%%%%%%%%%%%%%%%%%%
N = 1e9;  %set max number of bits for simulation
%%%%%%%%%%%%%%%%%%

SPB = round(12.525/STEPSIZE );%number of iterations per bit
%12.525 seconds is the bit period of a
%published work for comparison. Cuomo[93].

Eb_No_dB = 15:50;   %set desired points for comparison
Eb_Avg = 1335.5;    %Average chaotic bit energy from calcBitEnergy func
Eb_No = 10.^(Eb_No_dB./10);
No = Eb_Avg ./ Eb_No;  %these lines convert Eb/No to a std dev value.
sigma = sqrt(No ./ 2);
```

```
S = length(Eb_No_dB)  %nubmer of points on BER plot
BER = zeros(1,S);  %store results of BER simulation
BER_fid = fopen( 'BER_results.txt', 'w'  );%open file to save BER results

for s = 1:S  %loop for each value Eb/No simulated
    tic

    %set initial values
    V0 = [1; 1; 1];
    VF = V0;
    betterRF = V0;

    num_errors = 0; %counter to keep track of number of errors encountered
    n = 0;

    while ( (num_errors<200) & ( n <= N ))  %only run sim until 200 errors
        n= n + 1;

        %new message bit
        message = round(rand(1,1));  %create random binary message bit

        [XMitter VF] = binaryTransmitter(VF, message, STEPSIZE, SPB);

        noise = randn(1,Samples_per_Bit) * sigma(s);
%Gaussian noise with standard deviation sigma
        influence = XMitter + noise;%transmitted signal influence by noise

        [SysA AF] = receiverSysA(betterRF, influence, STEPSIZE, SPB);
        [SysB BF] = receiverSysB(betterRF, influence, STEPSIZE, SPB);

        %Received Bit determination
ErrorA = sum( ( influence - SysA ).^2 );
        ErrorB = sum( ( influence - SysB ).^2 );
        if (ErrorA > ErrorB)
            recovered_message = 1; %System B is the better match
            betterRF = BF;         %the last state of Rec Sys B is better
   %synced to the transmitter
        else
            recovered_message = 0; %System A is the better match
            betterRF = AF;         %the last state of Rec Sys A is better
   %synced to the transmitter
        end
```

```
        if(message - recovered_message) %test for error
            num_errors = num_errors + 1; %if message does not match
        end
    end

    t = toc;  %record time for this plot point on BER graph
    BER(s) = num_errors / n;  %errors per bit
    disp('----------------------------------------')
    % disp(['Average Error per bit: ', num2str(ERROR_AVG)] )
    disp(['Bit Error Rate (BER): ', num2str(BER(s)), ' at Eb/No: ',
 num2str(Eb_No_dB(s)), ' Bits Required: ', num2str(n),
 '  Time Required: ' , num2str(t), ' seconds.'] )
    disp('----------------------------------------')
    fprintf(BER_fid, '%1.7f   %2.1f   %9.0f   %7.2f\n',
 [BER(s), Eb_No_dB(s), n, t]);
    %save data in case of sim disaster...
    save BER1_results BER Eb_No_dB
end

fclose(BER_fid);
%plot BER curve
semilogy( Eb_No_dB, BER , 'x' )
title('Bit Error Rate')
xlabel('Eb/No (db)')
ylabel('BER')
grid on
```