Digital Reverse Time Chaos and Matched Filter Decoding

by

John Phillip Bailey III

A dissertation submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Doctor of Philosophy

Auburn, Alabama
May 10, 2015

Keywords: digital chaos, reverse time chaos, matched filter decoding

Approved by

Michael Hamilton, Chair, Assistant Professor of Electrical and Computer Engineering
Robert Dean, Associate Professor of Electrical and Computer Engineering
Lloyd Riggs, Professor of Electrical and Computer Engineering
Stuart Wentworth, Associate Professor of Electrical and Computer Engineering

Abstract

The use of reverse time chaos allows the realization of hardware chaotic systems that can operate at speeds equivalent to existing state of the art while requiring significantly less complex circuitry. Unlike traditional chaotic systems, which require significant analog hardware that is difficult to realize at high speed, the reverse time system can be realized with only a FPGA calculating a digital iterated map that drives a simple series RLC filter. Because the dynamics of this system are determined by an iterated map, both Lorenz-like and Rössler-like dynamics can be implemented without requiring any hardware adjustments. Precise control of this system can also be maintained by adjusting the initial condition of the iterated map.

Matched filter decoding is also possible for the reverse time system due to its possession of a closed form solution formed partially by a linear basis pulse. Coefficients have been calculated to realize the matched filter digitally as a FIR filter. Numerical simulations confirm that this correctly implements a matched filter that can be used for detection of the chaotic signal. In addition, the direct form of the FIR filter has been implemented in HDL with demonstrated performance in agreement with numerical results.

Acknowledgments

The author would like to express great appreciation and thanks to Dr. Michael Hamilton for serving as his advisor and providing significant guidance and assistance throughout the process of performing this work. Dr. Robert Dean, Dr. Lloyd Riggs, and Dr. Stuart Wentworth are also thanked for their review and commentary, as is Dr. Minseo Park for serving as the University Reader for this work.

Appreciation is expressed to Dr. Ned Corron for his discussions on the mathematics behind the oscillator and matched filter. The author would also like to thank Aubrey Beal and Keaton Rhea for their assistance in many aspects of this work.

A personal note of appreciation goes to John Bailey Jr. and Dawn Bailey for their support throughout the author's academic endeavors.

Contents

List of Figures

List of Tables

List of Abbreviations

ASIC	application specific integrated circuit
AWGN	additive white Gaussian noise
FFT	fast Fourier transform
FIR	finite impulse response
FPGA	field-programmable gate array
HBT	heterojunction bipolar transistor
HDL	hardware description language
IC	integrated circuit
LHS	left hand side
NIC	negative impedance converter
PCB	printed circuit board
PSD	power spectral density
RHS	right hand side
RLC	resistor-inductor-capacitor
SiGe	silicon germanium
SNR	signal-to-noise ratio

Chapter 1

Introduction

1.1 Overview

Although many types of chaos continue to evade analytic solutions, the assumption that no solvable chaotic dynamics exist has been broken by the discovery of hybrid (consisting of interdependent continuous time and discrete time states) chaotic systems which possess exact solutions [3–5]. A linear matched filter, which can be used to exactly recover the discrete-time state of the chaotic system when driven with only that system's continuous-time state, has also been developed using these solutions. Matched filter decoding significantly reduces the difficulty of integrating chaos into a wide range of electronic systems, including both communication and radar.

One of the most promising areas of research in chaotic systems has long been in their utility for communication [6, 7]. Previous work has shown that such systems can be constructed and tuned such that they possess advantageous dynamics for various communication schemes [8, 9]. In addition to possessing these targeted dynamics, chaotic systems can also be subjected to control schemes that can maintain desired trajectories [10]. By combining these characteristics with matched filter decoding [4, 11], many of the necessary components for a modern high-performance communication system may be realized with chaos-based systems.

Various elements of such systems have been analyzed to determine how they might perform in a number of applications. In the presence of additive Gaussian white noise, chaotic systems possess error rates comparable to communication systems currently in use [12, 13]. Multipath propagation has also been shown to not pose the significant challenge it usually presents when the propagated waveform is chaotic [14].

A second area of interest in recent chaotic research is in its application for radar waveform generation [15]. The inherently wideband nature of signals found in any chaotic system provides many advantages for radar [16, 17]. Using such techniques has the potential to significantly improve the resolution of radar systems while further reducing their susceptibility to intercept and jamming. In addition, both weather [18] and imaging [19] applications of radar systems can benefit from wideband chaotic signals.

Actually realizing physical systems that exhibit the chaotic dynamics necessary for these applications has long proven to be a difficult task. Although surprisingly simple systems constructed from both familiar [20, 21] and exotic [22, 23] components have been shown to behave chaotically, such systems do not readily lend themselves to either exact solutions or control. Simulations have shown that hardware can be developed for a controllable chaotic system with an exact solution [24], but this hardware relies on many analog components which are not expected to scale well to high speed operation.

Reverse time chaos provides a potential solution for realizing chaos in hardware with both solvable and controllable properties without sacrificing the ability to scale in speed. First proposed by Corron et. al. in [25], reverse time chaos describes behavior that differs from traditional chaos by using the current state of the system to represent all of its past states instead of all of its future states. Despite this difference, reverse time chaos retains a positive Lyapunov exponent and a corresponding sensitivity to initial conditions that defines traditional chaotic systems. The many similarities between traditional (or forward time) and reverse time chaos also allows for formulation of an exact solution. Because of this, in a method following the systems in [4, 5], the development of a linear matched filter which can be used to detect the original chaotic signal is possible.

1.2 Classical Chaotic Systems

To better understand the behavior of chaotic systems, it is helpful to first clearly define what properties such a system is expected to possess. Three significant characteristics are

often agreed upon as necessary for chaos: the system must exhibit aperiodic long-term behavior, it must be deterministic, and it must show sensitive dependence to its initial conditions [26, 27]. To meet the first criterion, the system must not develop any patterns (either long range or short range) in its behavior regardless of how long it is run. The second criterion requires that the system does not involve any random process. This leads to the conclusion that, provided all initial conditions are defined to infinite accuracy, the system will always produce the same output for a given input. A visual example, shown in the following figure, best describes the third criterion.

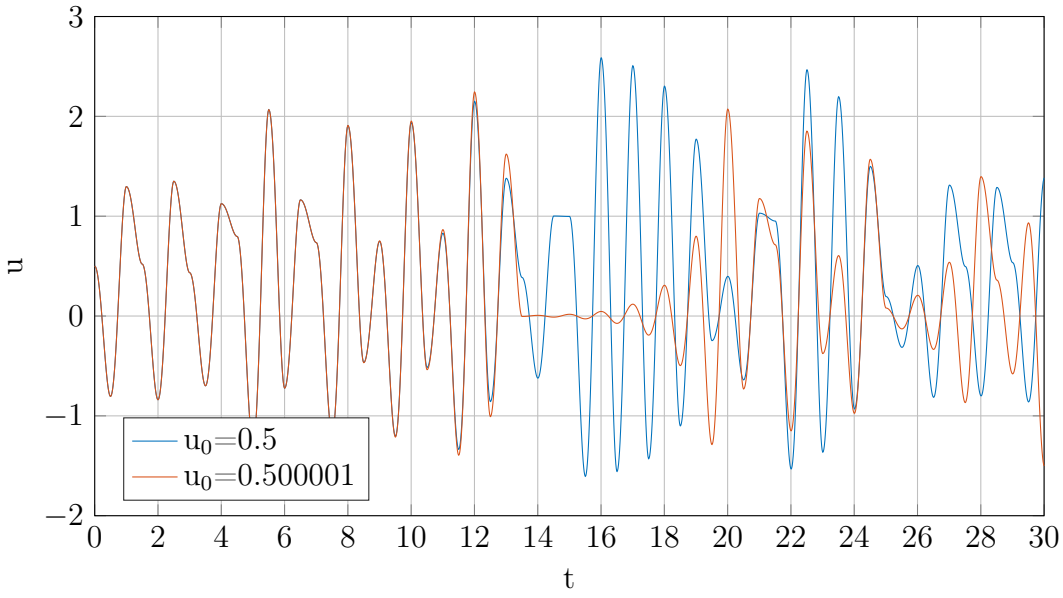

Figure 1.1: Two Chaotic Trajectories with Close Initial Conditions

Each curve shows a chaotic trajectory u starting at $t = 0$ with the initial condition u_0. The value of u_0 was changed by 0.000001 between the two runs (all other initial conditions remained constant). As shown, the trajectories of u begin to vary significantly after only a short time; this behavior demonstrates the requisite sensitivity to initial conditions. Note that u and t are both plotted as dimensionless quantities - this convention will be followed for all results generated using numerical models.

It is important to highlight the caveat requiring initial conditions be defined to infinite accuracy in the second criterion for chaos, because, in any physical system or any numerical model not using fixed width representations, such a definition is not achievable. When combined with the conclusions from the third criterion for chaos, this caveat gives rise to the seemingly random behavior shown by chaotic systems.

1.2.1 Lorenz

The canonical example of a chaotic system is often given by the Lorenz system, first described by Edward Lorenz in 1963 as a rudimentary means of atmospheric modelling. Prior to this work, it was a commonly held belief that low order systems were incapable of generating complex dynamics. The Lorenz system can be described by the following three coupled equations:

$$\dot{x} = -\sigma x + \sigma y \tag{1.1}$$

$$\dot{y} = -xz + rx - y \tag{1.2}$$

$$\dot{z} = xy - bz \tag{1.3}$$

where x, y, and z are the state variables and σ, r, and b are control parameters [28]. Although the mathematical description of this system appears relatively simple (only two nonlinear terms are present), certain sets of control parameters will produce highly complex chaotic dynamics. Figure 1.2 shows a chaotic trajectory for x generated by a numerical simulation with the values $\sigma = 10$, $r = 28$, and $b = \frac{8}{3}$.

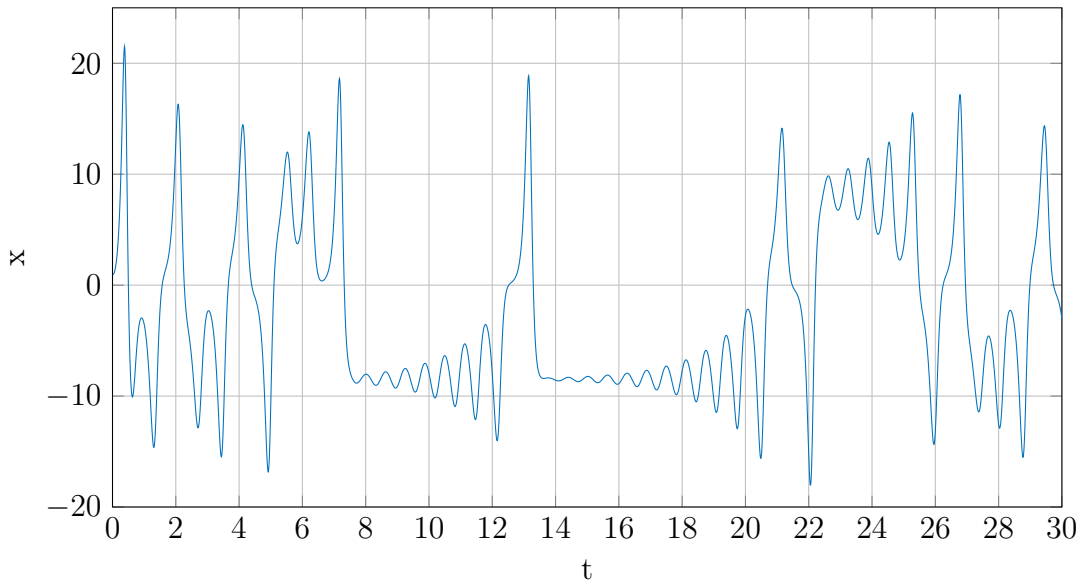

Figure 1.2: Numerical Simulation of Lorenz System (time)

The dynamics of x demonstrate significant complexity when compared against its relatively simple governing equations. Although x appears to follow a pattern of oscillating around two distinct levels, the number of oscillations which occur after each switch does not (and will never) settle in to any period. This type of behavior, where the time around each level can take any value, is often referred to as Lorenz like dynamics.

Often referred to as a phase portrait, a plot of one of a chaotic system's state variables against one or more of its other state variables is a common method to gain additional insight into the dynamics of chaotic systems. Eliminating time as a variable allows for long range dynamics to become apparent and provides details not otherwise visible. A phase portrait for the Lorenz system with y and z plotted versus x is shown in the following figure.

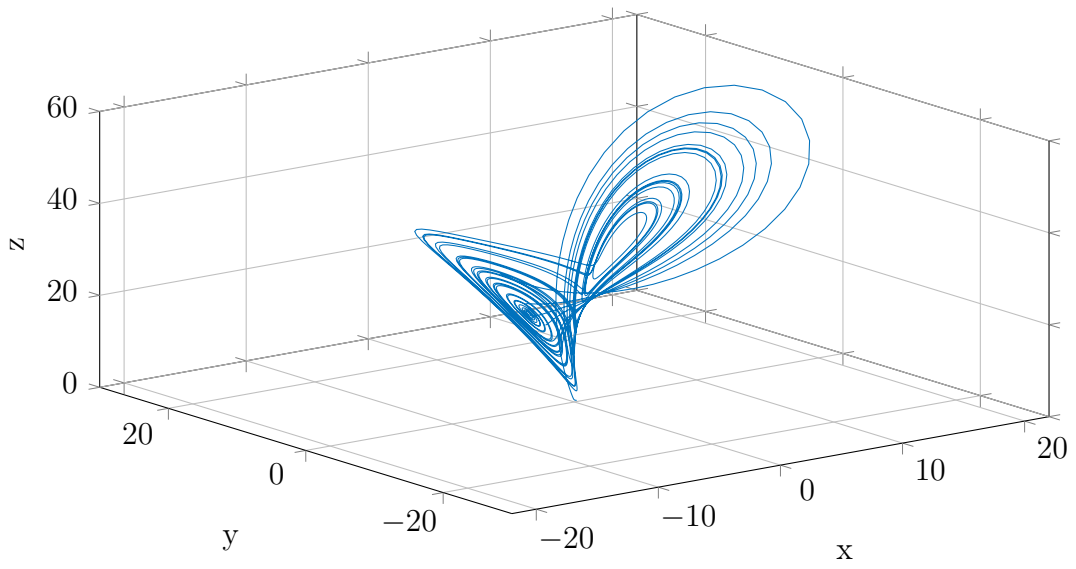

Figure 1.3: Numerical Simulation of Lorenz System (phase)

The shape shown in Figure 1.3 is commonly known as the Lorenz Butterfly. As in the plot versus time, visual inspection reveals two points around which the trajectories appear to orbit with differing values in the radius of these orbits corresponding to the varying amounts of time elapsed around each level. Additional symbolic analysis would reveal that these points are the singular points of the system. Not shown is the amount of time elapsed; for Figure 1.3, the simulation was stopped at $t = 1000$. This extended simulation time reveals that the system experiences long term boundedness. The bounding area in which the trajectories of the system remains is often referred to as an attractor.

1.2.2 Rössler

A second well known chaotic system was originally developed in 1976 by Otto Rössler. This system was not developed as a means to describe a physical phenomenon, but rather as an attempt to produce a model with dynamics similar to the Lorenz system without the

difficulty in analysis. The Rössler system is defined as:

$$\dot{x} = -y - z \tag{1.4}$$

$$\dot{y} = x + ay \tag{1.5}$$

$$\dot{z} = b + zx - zc \tag{1.6}$$

where x, y, and z are again the state variables and a, b, and c are control parameters [29]. This system reduces analytic complexity by possessing only a single nonlinear term, but, as with the Lorenz system, will generate chaotic trajectories for some sets of control parameters. The trajectory of x is shown in the following figure for $a = 0.1$, $b = 0.1$, and $c = 14$.

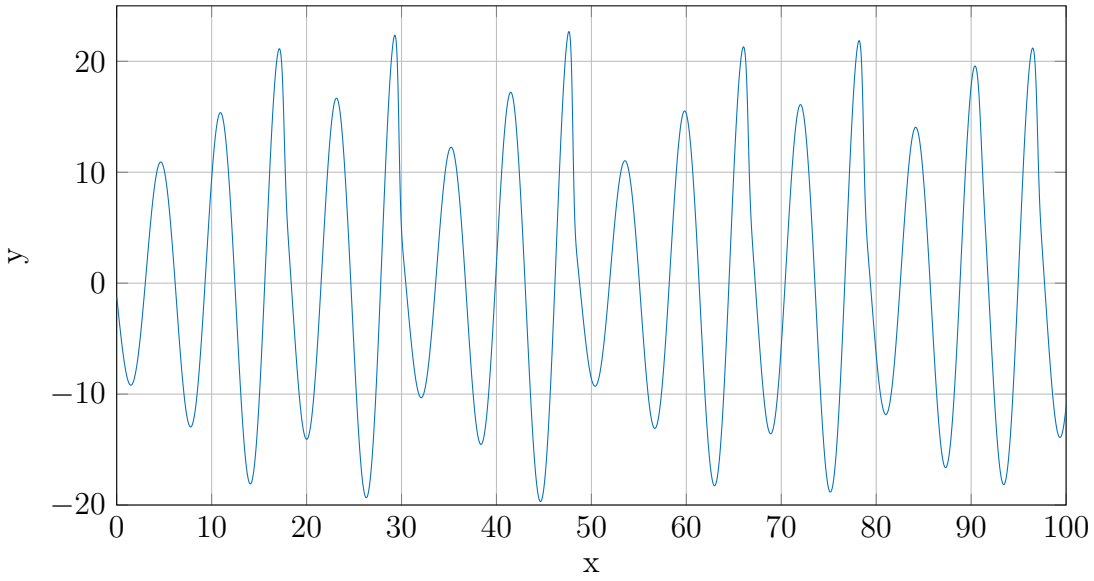

Figure 1.4: Numerical Simulation of Rössler System (time)

At first glance, the Rössler system does not appear to mimic any of the complex behavior readily apparent in the Lorenz system. Closer inspection reveals that the magnitude of x grows for varying durations of t before abruptly decreasing. The phase portrait, shown below in Figure 1.5, provides a clearer view of this process.

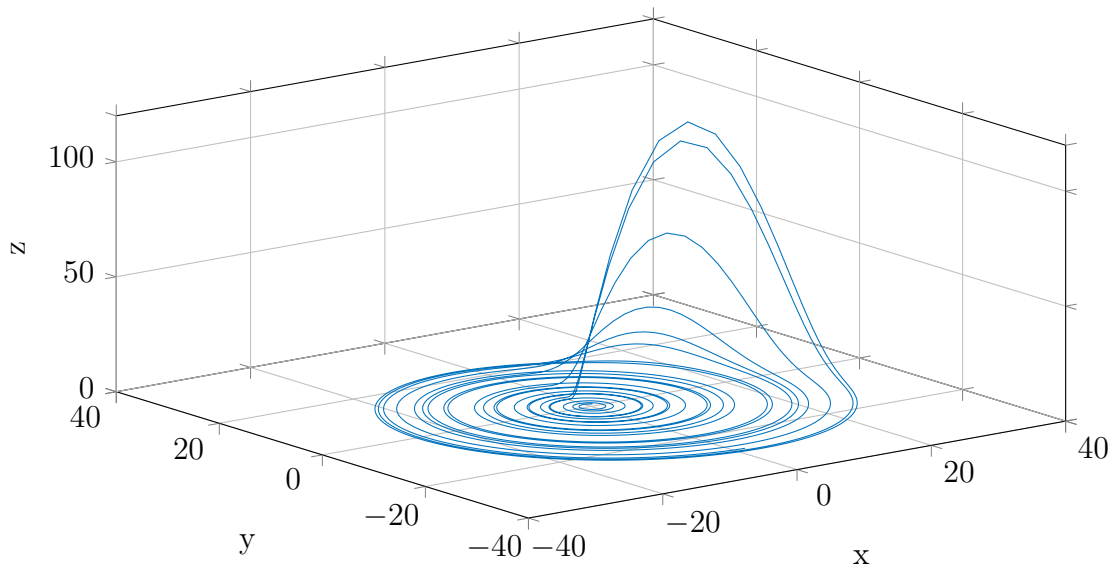

Figure 1.5: Numerical Simulation of Rössler System (phase space)

Most of the orbits in this system remain centered around a single point, explaining the almost sinusoidal appearance in Figure 1.4. Occasionally, however, an orbit will jump up in the z plane; when this event occurs, it never lasts for more than one-half of an orbit. This type of behavior is referred to as a fold and serves as the defining characteristic of the Rössler system.

1.2.3 Chua

While it does not generate its own distinct type of chaos, Chua's circuit (named after Leon Chua) provides an excellent introduction to the development of electronic systems that can mimic chaotic dynamics described by mathematical models. The governing equations for this system are given by:

$$\dot{x} = \alpha x - \alpha y - \alpha f(x) \tag{1.7}$$

$$\dot{y} = x - y + z \tag{1.8}$$

$$\dot{z} = -\beta y \tag{1.9}$$

where x, y, and z are the state variables, α and β are control parameters, and $f(x)$ is a function that provides a nonlinear response [30]. A schematic is shown in Figure 1.6 for a circuit implementation of Equations 1.7-1.9.

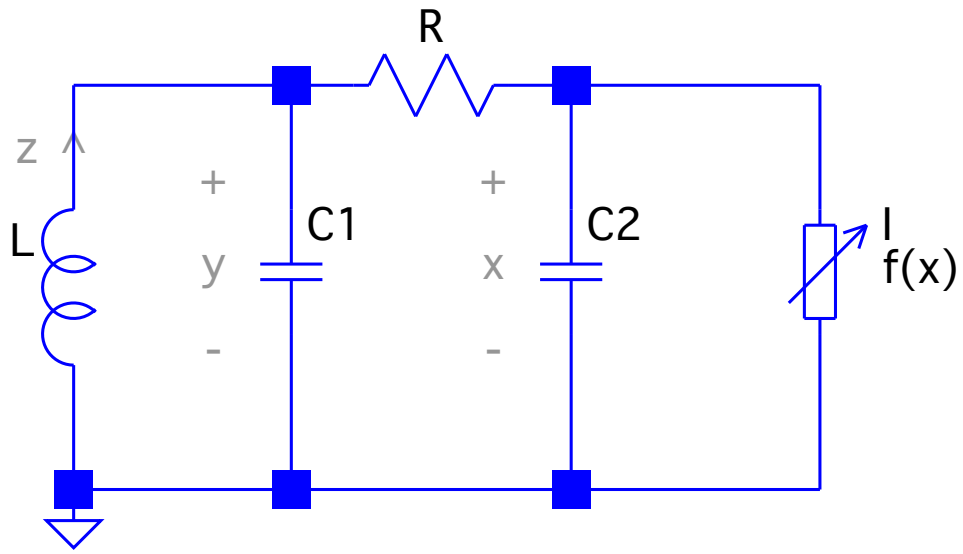

Figure 1.6: Chua's Circuit Schematic

In this circuit, x represents the voltage across capacitor `C2`, y the voltage across capacitor `C1`, and z the current through inductor `L`. An extra resistor $\texttt{R}_\texttt{S}$ in series with `L` models the resistance of the inductor's windings (not shown). Equations 1.7-1.9 can be rewritten in terms of these circuit elements as:

$$\dot{x} = \frac{1}{C_2}\left(\frac{1}{R}(y - x) - f(x)\right) \tag{1.10}$$

$$\dot{y} = \frac{1}{C_1}\left(\frac{1}{R}(x - y) + z\right) \tag{1.11}$$

$$\dot{z} = \frac{1}{L}(y + R_S z) \tag{1.12}$$

Realizing $f(x)$ often requires a separate circuit; the Chua diode is commonly used for this function [1]. Figure 1.7 shows the I-V characteristic of a Chua diode with two slopes.

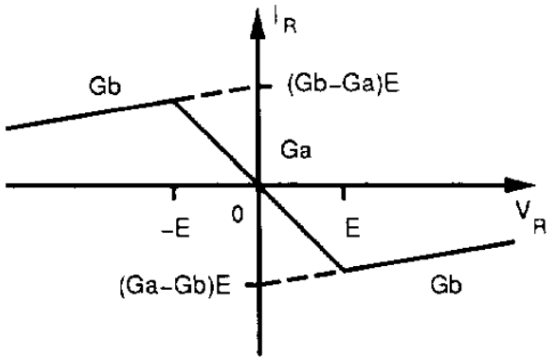

Figure 1.7: I-V Curve of the Chua Diode [1]

Each point where the slope abruptly changes in this response results in a new singular point around which the dynamics of the Chua circuit will oscillate. By varying the number and value of slopes present in the response, differing dynamics can be generated without requiring any additional changes to the circuit [31]. The following figures demonstrate this graphically.

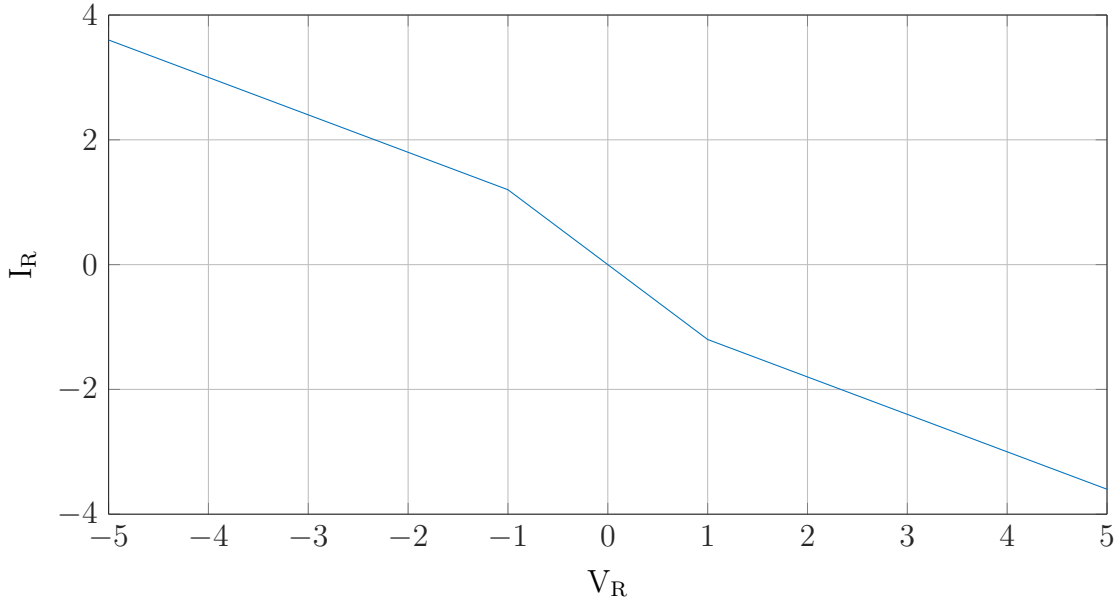

Figure 1.8: $f(x)$ Transfer Characteristic

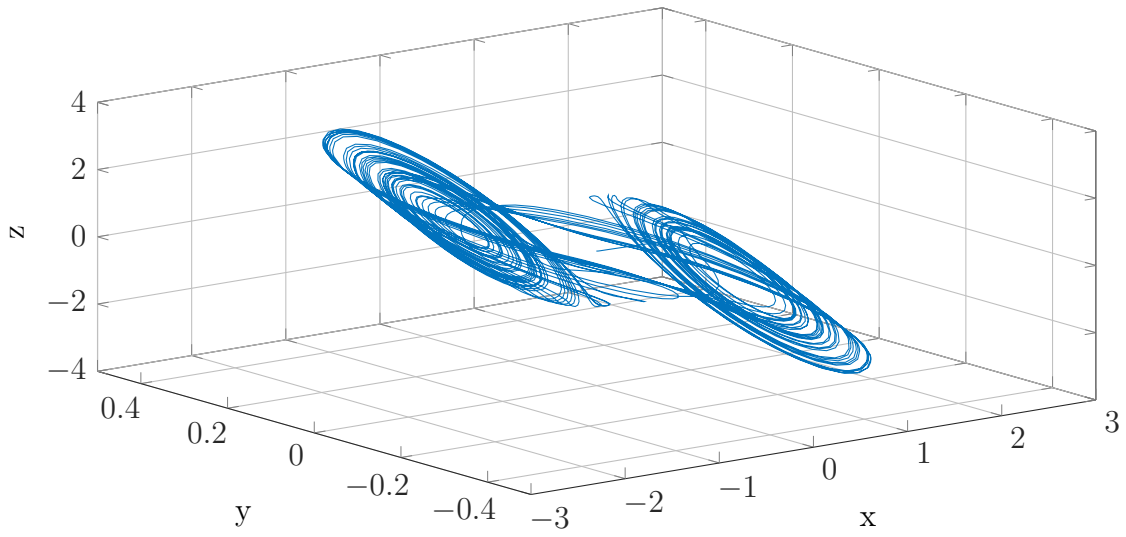

Figure 1.9: Numerical Simulation of Chua's Circuit (phase portrait)

In Figure 1.8, two negative slopes are used to define $f(x)$; the values of these were set to -0.6 and -1.2. The phase portrait (Figure 1.9) displays an attractor with two distinct singular points as expected. Increasing the magnitude of the slopes causes the attractor to expand in all directions. This behavior is shown in Figures 1.10 and 1.11.

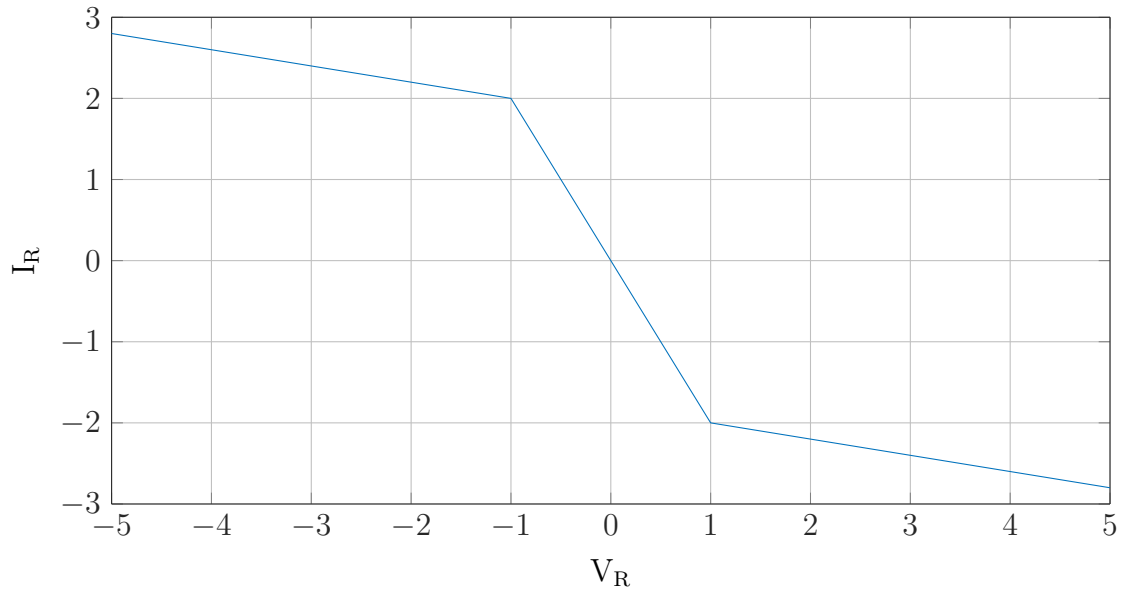

Figure 1.10: $f(x)$ Transfer Characteristic

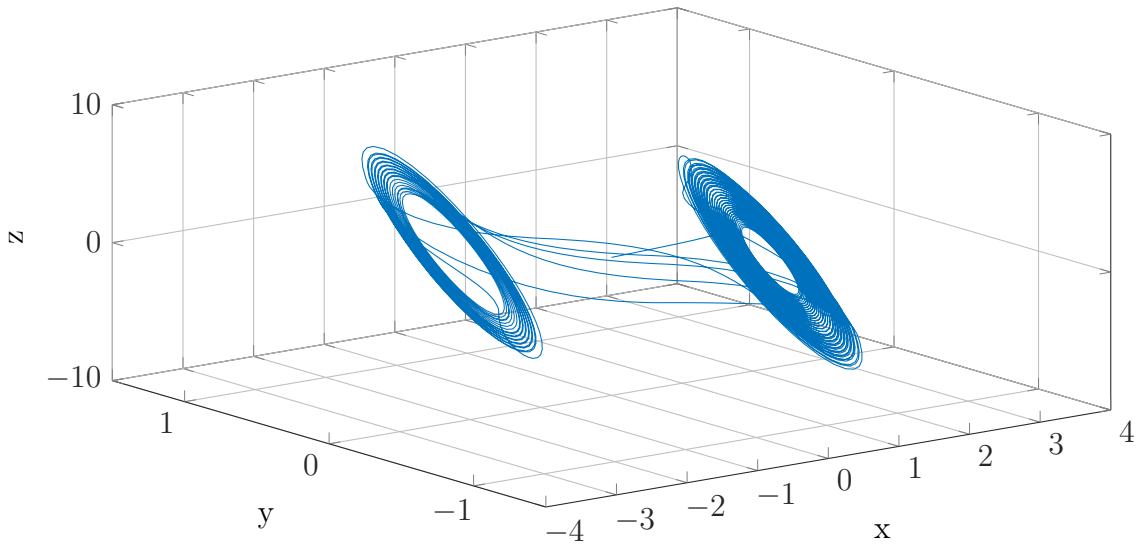

Figure 1.11: Numerical Simulation of Chua's Circuit (phase portrait)

The ability to adjust the system to exhibit multiple types of dynamics by simply changing the properties of a single circuit element makes Chua's circuit an often used tool for exploring many types of chaotic behavior. Due to its low number of required elements, prototypes of the circuit can be constructed successfully on a breadboard; this same simplicity allows scaling all the way to full IC designs [32].

1.3 Exactly Solvable Chaos

1.3.1 Theory

Although all of the systems discussed in the previous sections can be expressed in relatively simple mathematical terms, none of them have (at the time of this writing) known exact solutions. Numerical simulation provides extensive insight in to the dynamics of these systems but does not allow for the same type analysis as is possible with a closed form solution. First described in [4], an exactly solvable system has been developed that can exhibit chaotic behavior for a range of control parameter values.

This chaotic system can be described by a set of two equations written in terms of two states. These consist of a continuous time state $u(t) \in \mathbb{R}$ which provides exponential growth so that the system does not die out and a discrete time state $s(t) \in \{-1, 1\}$ or $s(t) \in \{0, 1\}$ which maintains boundedness so that the system does not blow up. $u(t)$ is defined by the continuous-time differential equation:

$$\ddot{u} - 2\beta\dot{u} + (\omega^2 + \beta^2) * (u - s) = 0 \tag{1.13}$$

where $\omega = 2\pi$ and $0 < \beta \leq \ln 2$. Taken alone, this expression is a simple second order linear differential equation with negative damping, which satisfies the exponential growth requirement for $u(t)$. A guard condition is then enforced by the definition of $s(t)$ to maintain the necessary bounds. This is accomplished by limiting the evaluation of $s(t)$ to only the instants when the derivative of $u(t)$ is zero (every $\frac{1}{2}t$). In a manner similar to Chua's circuit, $s(t)$ is defined as a nonlinear function which is then included in an otherwise linear system to induce chaotic behavior. Multiple dynamics can be realized by selecting specific definitions of $s(t)$; for Lorenz like dynamics, $s(t)$ is defined as:

$$\dot{u} = 0 \rightarrow s(t) = sgn(u(t)) \tag{1.14}$$

where

$$sgn(u) = \begin{cases} -1 & u < 0 \\ 1 & u \geq 0 \end{cases} \tag{1.15}$$

When operating with this $s(t)$, the system is said to be in the shift band. Redefining $s(t)$ to generate Rössler like dynamics yields:

$$\dot{u} = 0 \rightarrow s(t) = H(u(t)) \tag{1.16}$$

where

$$H(u) = \begin{cases} 0 & u \leq 0 \\ 1 & u > 0 \end{cases} \tag{1.17}$$

The system is said to be in the folded band when Equation 1.17 is used to define $s(t)$. For both bands, an oscillation represented by $u(t)$ initially grows exponentially about the value of the discrete state $s(t)$, but is bounded when it exceeds the guard condition. At this point, a transition is triggered in $s(t)$ squelching the oscillation and causing its center to shift to the other discrete value of $s(t)$; this process then repeats continually. Figures 1.12 and 1.13 demonstrate this behavior graphically for the shift band and folded band, respectively.

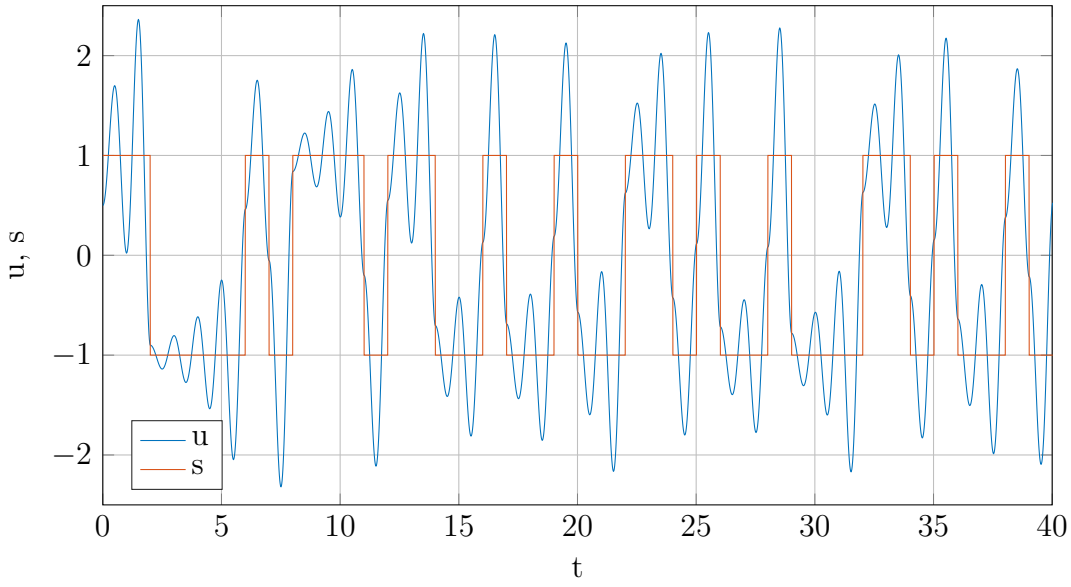

Figure 1.12: Numerical Simulation of Exactly Solvable System in Shift Band (time)

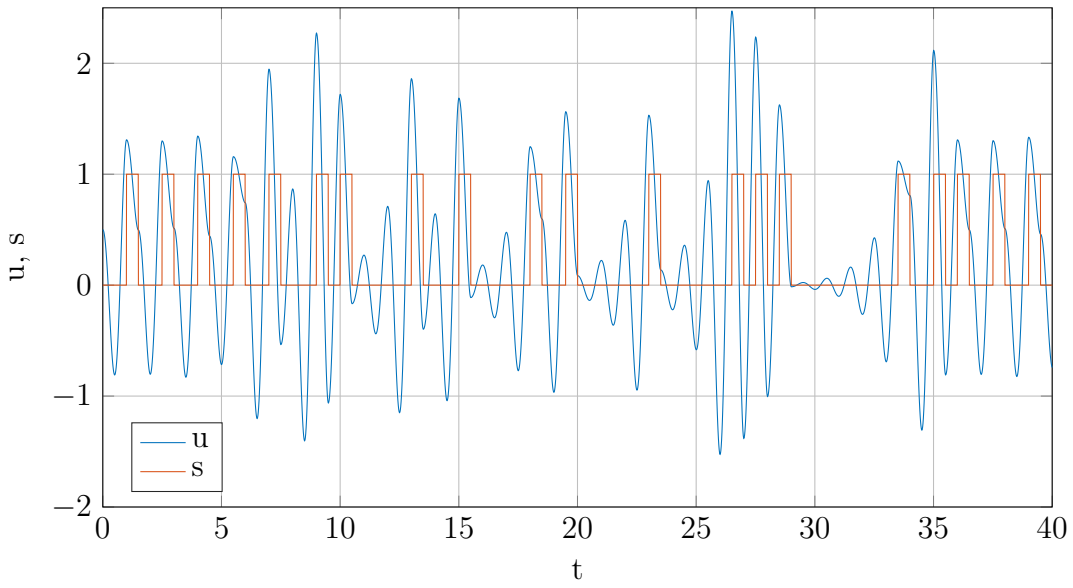

Figure 1.13: Numerical Simulation of Exactly Solvable System in Folded Band (time)

As with the previously discussed chaotic systems, the phase portraits of this system operating in each band provide more obvious insight into long term dynamics. The phase portrait for the shift band is presented in the following figure.

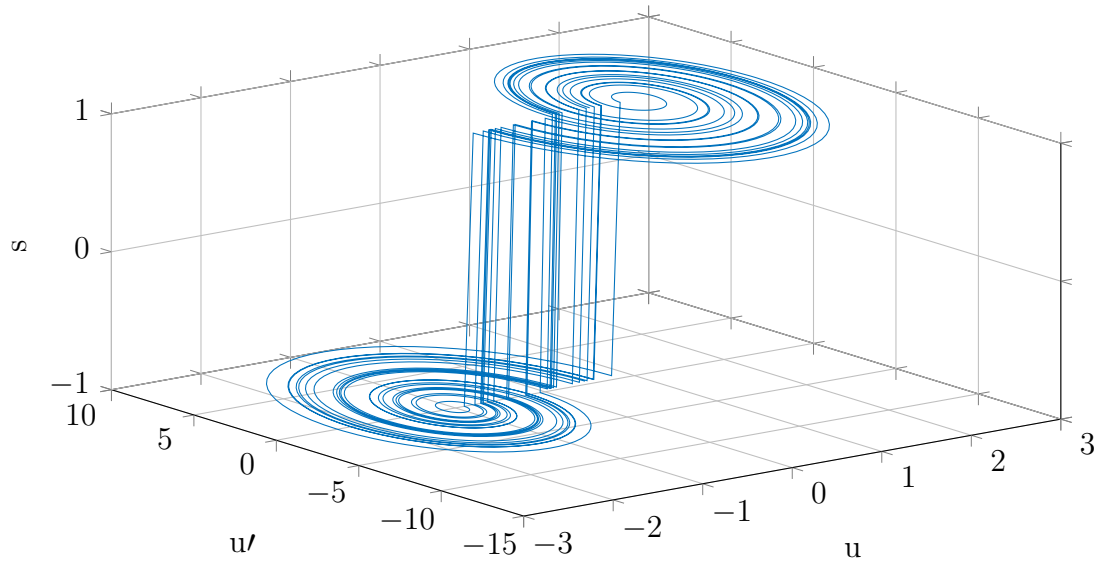

Figure 1.14: Numerical Simulation of Exactly Solvable System in Shift Band (phase portrait)

In this plot, u is plotted on the x axis, $u\prime$ (equivalent notation to $\dot{u}$ and $\frac{\mathrm{d}u}{\mathrm{d}t}$) on the y axis, and s on the z axis. This allows for the effect of a transition in s to be seen as a shift in the level around which the trajectories orbit. Also visible is the property that the system will always complete at least a single full orbit at its current s level before a switch in s occurs. In contrast, when operating in the folded band, the system does not orbit each of its s values equally. This is shown in Figure 1.15.

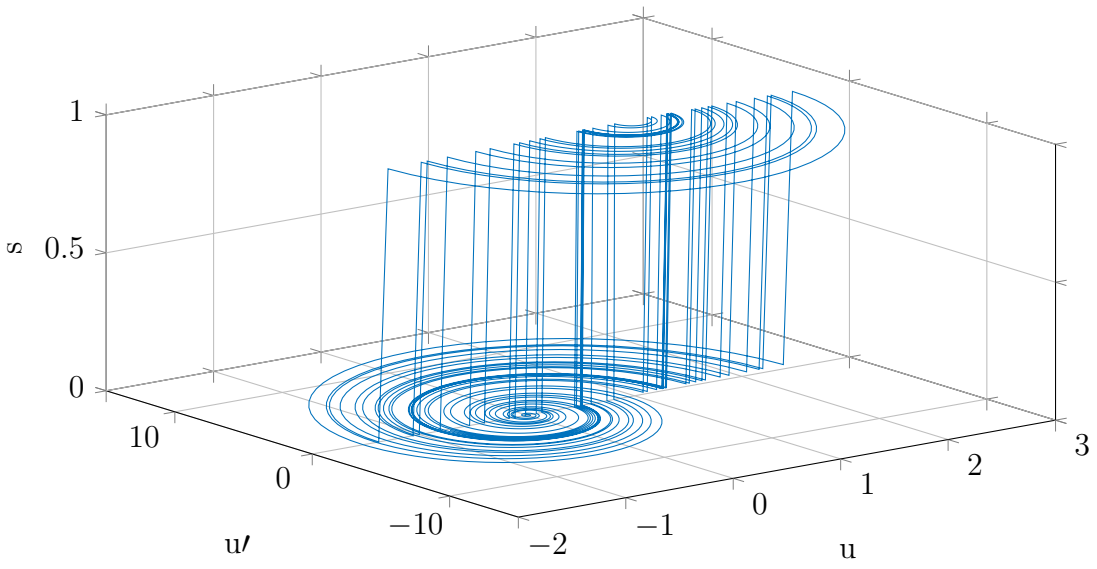

Figure 1.15: Numerical Simulation of Exactly Solvable System in Folded Band (phase portrait)

Here, the system only spends one-half of an orbit around the higher level of s before switching back down to the lower level. This is similar in behavior to the Rössler system and gives rise to the folded band designation for this mode of operation. While not immediately visible in either the time or phase portrait plots, the timing of the system also changes in this band. Additional details on this phenomenon are discussed in the section on chaotic maps.

An analytic solution to Equation 1.13 is given in [4] as:

$$u(t) = \sum_{m=-\infty}^{\infty} s_m P(t-m) \tag{1.18}$$

$$P(t) = \begin{cases} (1-e^{-\beta})e^{\beta t}(\cos(\omega t) - \frac{\beta}{\omega}\sin(\omega t)) & t < 0 \\ (1-e^{\beta(t-1)})(\cos(\omega t) - \frac{\beta}{\omega}\sin(\omega t)) & t = 0 \\ 0 & t > 0 \end{cases} \tag{1.19}$$

where $P(t)$ acts as a basis function. Although the simple existence of this solution is remarkable, its most significant characteristic is in its description in Equation 1.18 in terms of convolution of a basis function with a random bipolar pulse. This description notably allows for derivation of a matched filter that can be used to partially regenerate $s(t)$ when presented with only $u(t)$. From [33], this filter can be described by:

$$\ddot{\xi} + 2\beta\dot{\xi} + (\omega^2 + \beta^2)\xi = (\omega^2 + \beta^2)\eta(t) \tag{1.20}$$

where

$$\dot{\eta} = v(t+1) - v(t) \tag{1.21}$$

In Equations 1.20 and 1.21, ξ represents the output of the matched filter (which is also the recovered $s(t)$) while $v(t)$ represents the input, which is assumed to be $u(t)$.

1.3.2 Chaotic Oscillator

A simulation model for an electronic circuit has been presented in [24] operating at a frequency of approximately 1.6 MHz. While the circuit initially appears complex, it can be broken into three major blocks for simplified analysis. These three blocks are indicated visually on the schematic included on the next page.

Basis Pulse

An exponentially growing oscillation described by the basis function is provided by the first block, highlighted in red. This is accomplished by replacing the R component typically used in a parallel RLC tank with a NIC . This subcircuit mimics the behavior of a negative resistor and provides the energy necessary for the oscillation to grow rather than decay. While the exact value of the negative impedance varies as a function of frequency, it is approximately equal to `R3` at low frequencies.

Differentiator

The second block, highlighted in green, generates the derivative of u. This term is necessary for evaluation of the guard condition. In order to slightly simplify the circuit, the capacitor used by the RLC tank is also used in the differentiator (`C2` on the schematic).

Guard Condition

Most of the circuit consists of the third block, which acts as the guard condition and is highlighted in blue. The comparators `U3` and `U4` generate logic levels that are then fed into the logic blocks to result in an output to be fed back to the tank. The following truth table describes the logic block's functionality.

A	**B**	**U7_O**	**U10_O**	**U9_O**	**U8_O**	**S**
L	L	L	L	U8_O	L	L
L	H	L	H	U8_O	U9_O	U9_O
H	L	L	H	U8_O	U9_O	U9_O
H	H	H	H	H	U9_O	H

Table 1.1: Logic Block Truth Table

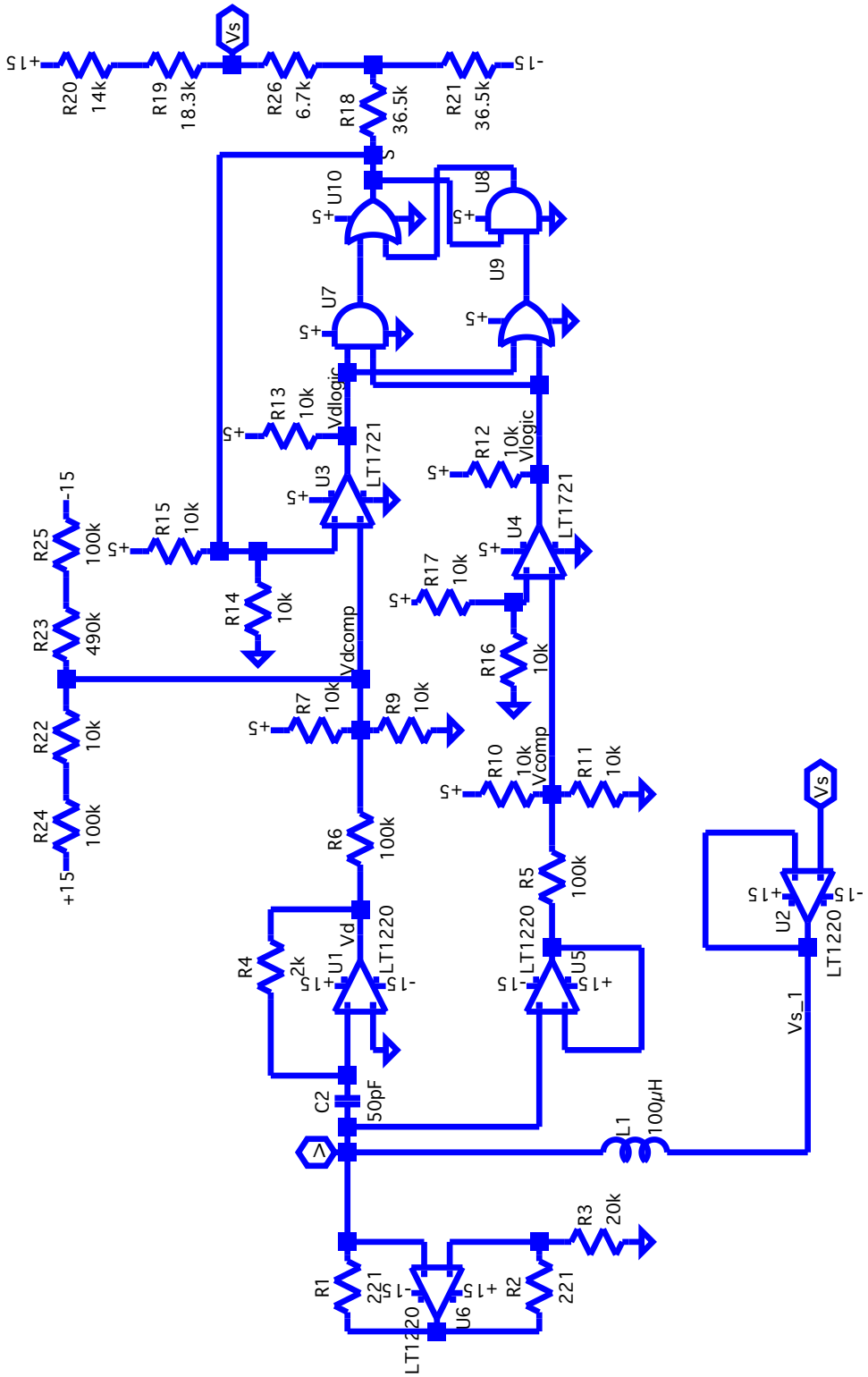

Figure 1.16: Original Chaotic Circuit Schematic

1.3.3 Matched Filter

A matched filter has also been demonstrated in simulation [34] operating in the audio range (approximately 2 kHz). Unlike the chaotic oscillator, which is free running, the matched filter requires two inputs to account for the forcing function in Equation 1.20. These inputs are the received signal `Vin` and a time delayed version of the received signal `Vdelay`. As shown in [4], the time delay required can be determined by:

$$T = 2RC\omega\sqrt{\frac{L}{4R^2C - L}} \tag{1.22}$$

where R, L, and C are the values used in the matched filter's tank. The schematic included on the following page does not include a block for generating `Vdelay`.

Both inputs are fed into a difference amplifier to implement Equation 1.21. This output is then integrated and used to drive an RLC tank circuit. In this tank, the resistor value is positive and results in the exponentially decaying oscillation necessary for the matched filter operation. The values for `L2`, `R38`, and `C3` correspond to the |R|, L, and C values used in the chaotic oscillator.

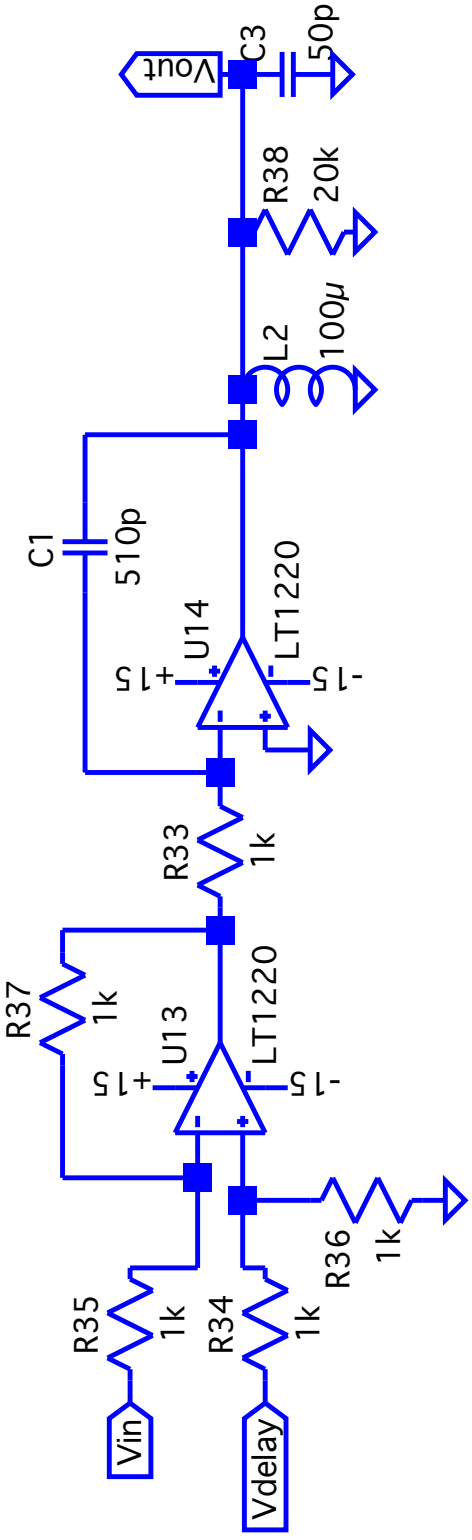

Figure 1.17: Original Chaotic Matched Filter Schematic

Chapter 2

Reverse Time Chaos

2.1 Motivation

Although both components of the forward time exactly solvable system have been demonstrated in circuit simulations, these circuits perform at relatively low speeds. Each circuit consists of many analog components that scale poorly with frequency and do not integrate well with modern high density IC processes. In particular, the use of a negative impedance converter provides a significant challenge for high frequency designs of the oscillator while the requirement for a long delay time adds significant complexity to the matched filter design.

Reverse time chaos provides a method for generating functionally equivalent waveforms with significantly simpler circuit implementations. The reverse time chaotic oscillator requires only three passive components (forming a series RLC tank) in its simplest form; the remainder of the circuit can be implemented digitally. Complementing this, the reverse time matched filter can be implemented entirely in digital circuitry.

2.2 Theory

To begin, a mathematical description of reverse time chaos can be defined by the second order system:

$$\ddot{u} + 2\beta\dot{u} + (\omega^2 + \beta^2)u = s(t) \tag{2.1}$$

as first described in [25]. Both β and ω are control parameters for this system; however, to maintain consistency with typical engineering practice, ω will be defined as the radian

frequency of operation. $s(t)$, the forcing function, can be described as

$$s(t) = As_n \tag{2.2}$$

where $s_n \in \{-1, 1\}$ is a random sequence with amplitude A. For the remainder of this work, the assumption that $A = 1$ will be made, but, in general, A can take any real value.

Upon inspection, it is apparent that Equation 2.1 describes a harmonic oscillator with positive damping producing oscillations centered around the value of $s(t)$. An illustration of this behavior generated with numerical simulation for $\beta = \ln(2)$ and $\omega = 2\pi$ is shown in Figure 2.1.

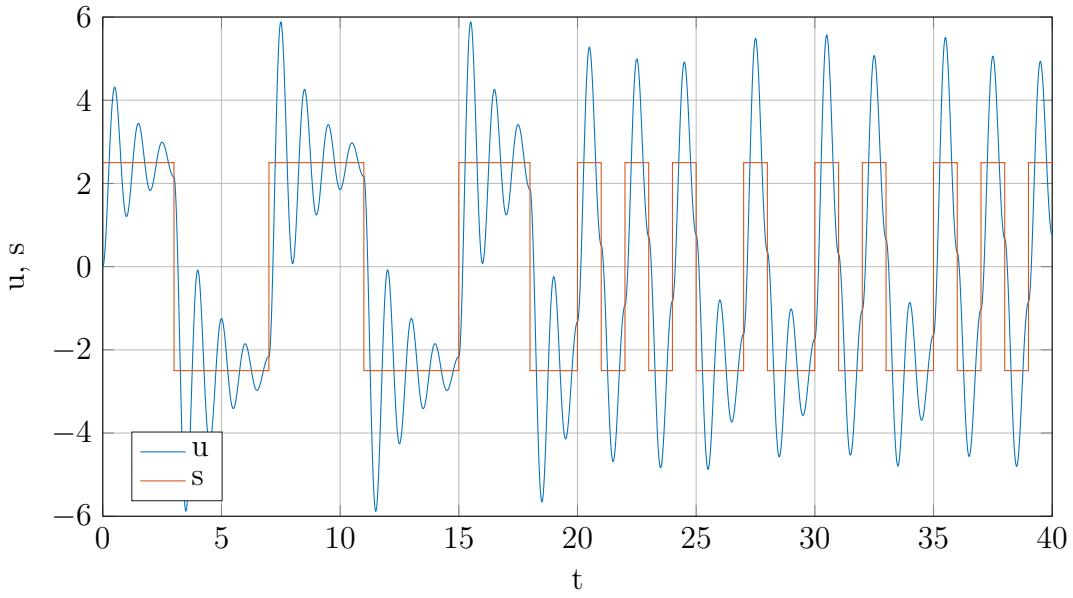

Figure 2.1: Numerical Simulation of Reverse Time System in Shift Band (time)

The negative damping term causes the value of $u(t)$ to decrease in magnitude and is the reason that the reverse time system requires $s(t)$ to act as a forcing function. Without this forcing, $u(t)$ would simply die out after ringing down to 0.

As described in the introduction, a reverse time chaotic system differs from a forward time chaotic system by using its initial condition to describe the set containing all of its

previous states rather than its future states. To better visualize this, Figure 2.2 below shows Figure 2.1 with its x axis reversed.

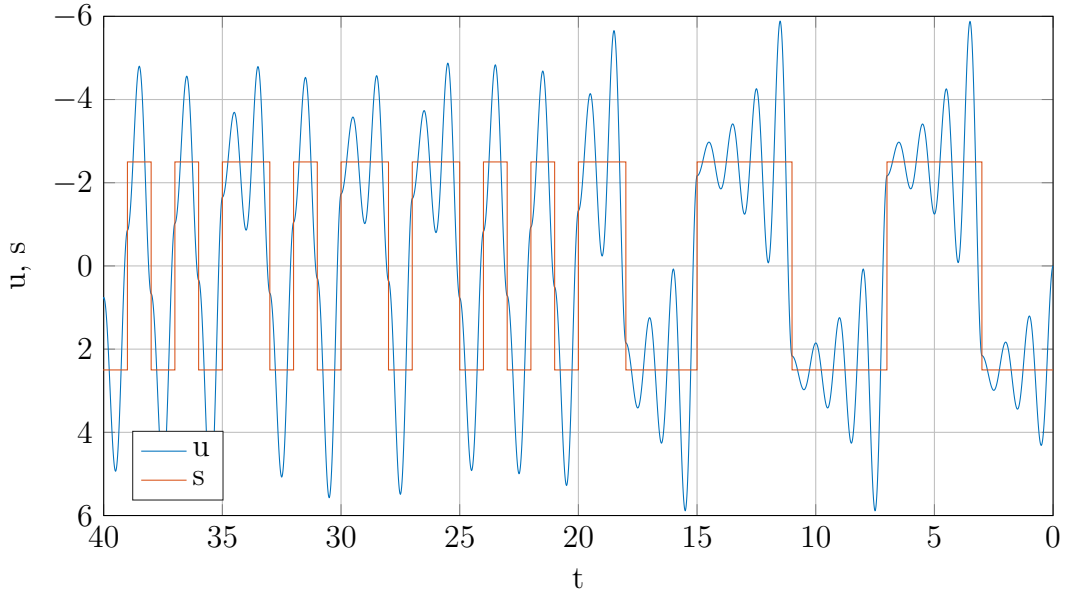

Figure 2.2: Numerical Simulation of Reverse Time System in Shift Band (reverse time)

When viewed in this manner, the dynamics of $u(t)$ appear identical to those of 1.13 operating in the shift band. Further confirmation of the similarity is shown by examining the phase space projection, shown in Figure 2.3.

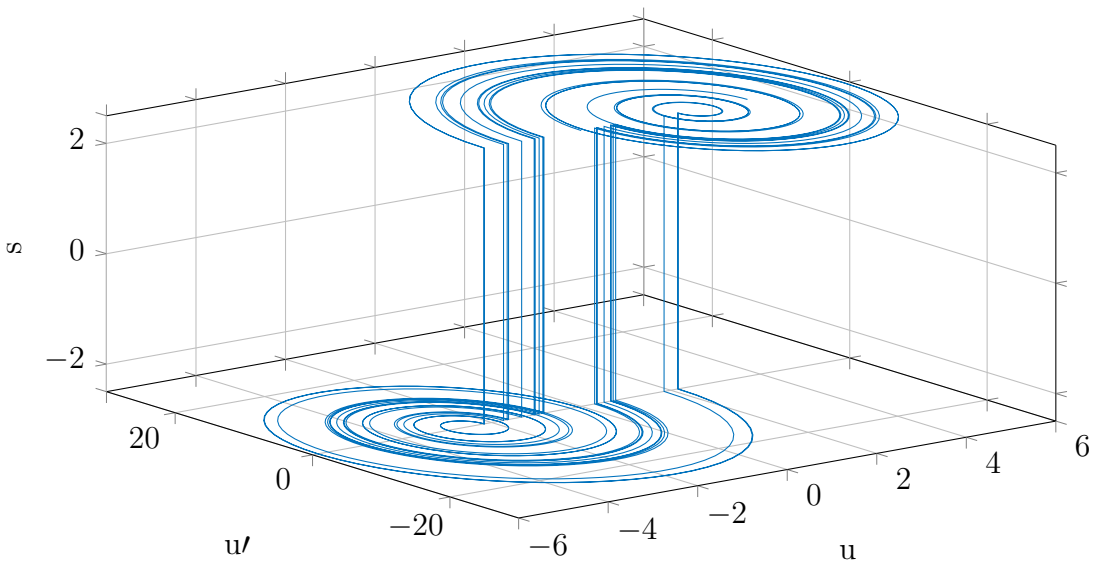

Figure 2.3: Numerical Simulation of Reverse Time System in Shift Band (phase portrait)

The reverse time system's phase portrait matches almost exactly with the phase portrait from the exactly solvable system. A notable difference is the levels of s around which the trajectories orbit; in the previous chapter, these were -1 and 1, while in the above figure they are -2.5 and 2.5. This discrepancy arises from scaling necessary in the numerical analysis routine used for the reverse time system. Implementing this system required a complex routine which constructs piecewise solutions between zero crossings of $\dot{u}$ and then combines these to generate the total solution. MATLAB code and additional details are provided in the appendices.

Solution

An exact solution for Equation 2.1 can be found, further continuing the similarity between the reverse time and exactly solvable systems. To begin the process, its homogeneous solution $u_h(t)$ must first be found. This can be done by setting the forcing function $s(t)$ equal to 0. With this substitution, Equation 2.1 becomes:

$$\ddot{u} + 2\beta\dot{u} + (\omega^2 + \beta^2)u = 0 \tag{2.3}$$

The characteristic equation for Equation 2.3 can be determined by inspection to equal:

$$\alpha\ddot{u} + \gamma\dot{u} + \delta u = 0 \tag{2.4}$$

Eigenvalues can now be found by substituting for u with λ raised to the power equal to the order of the derivative of u:

$$\alpha\lambda^2 + \gamma\lambda + \delta = 0 \tag{2.5}$$

Solving for λ in Equation 2.5 which results in:

$$\lambda = \beta \pm j\omega \tag{2.6}$$

As anticipated for a second order harmonic oscillator, these eigenvalues form a complex conjugate. The homogeneous solution can be written directly as:

$$u_h(t) = C_1 e^{-t(\beta+j\omega)} + C_2 e^{-t(\beta-j\omega)} \tag{2.7}$$

Simplifying the exponentials:

$$u_h(t) = C_1 e^{-\beta t} e^{-j\omega t} + C_2 e^{-\beta t} e^{j\omega t} \tag{2.8}$$

where C_1 and C_2 are constants of integration and $\omega \equiv 2\pi f$. Recall Euler's formula:

$$e^{\pm j\omega x} = \cos(\omega x) \pm j\sin(\omega x) \tag{2.9}$$

With this, equation 2.8 can be rewritten as:

$$u_h(t) = e^{-\beta t}(A\cos(\omega t) + B\sin(\omega t)) \tag{2.10}$$

In standard form, this reduces to:

$$u_h(t) = \alpha e^{-\beta t} \sin(\omega t + \phi) \tag{2.11}$$

where A and B (or α and ϕ) can be real or complex valued. To find values for A and B, initial values can be established by setting $t = 0$. Because there are two constants, two equations will be required; $\dot{u_h}(t)$ serves as the second equation. To find $\dot{u_h}(t)$, the form in Equation 2.10 is used:

$$u_h(t) = e^{-\beta t}(A\cos(\omega t) + B\sin(\omega t)) \tag{2.12}$$

Because $u_h(t)$ consists of a a product of two terms, the product rule must be used to find its derivative:

$$(\dot{fg}) = \dot{f}g + \dot{g}f \tag{2.13}$$

This results in:

$$\dot{u_h}(t) = e^{-\beta t}(B\omega\cos(\omega t) - A\omega\sin(\omega t)) - \beta e^{-\beta t}(A\cos(\omega t) + B\sin(\omega t)) \tag{2.14}$$

Evaluating at $t = 0$:

$$u_h(0) = e^{-\beta * 0}(A\cos(\omega * 0) + B\sin(\omega * 0)) \tag{2.15}$$

$$\dot{u_h}(0) = e^{-\beta * 0}(B\omega\cos(\omega * 0) - A\omega\sin(\omega * 0)) - \beta e^{-\beta * 0}(A\cos(\omega * 0) + B\sin(\omega * 0)) \tag{2.16}$$

Equations 2.15 and 2.16 can be simplified by recalling that $cos(0) = 1$ and $sin(0) = 0$. Making these simplifications leaves:

$$u_h(0) = A\dot{u_h}(0) = B\omega - \beta A \tag{2.17}$$

Solving for A and B:

$$\dot{u_h}(0) = B\omega - \beta u(0) \tag{2.18}$$

$$B = \frac{\dot{u}(0)}{\omega} - \frac{\beta u(0)}{\omega} \tag{2.19}$$

$$A = u_h(0) \tag{2.20}$$

A and B can now be substituted back into Equation 2.10:

$$u_h(t) = e^{-\beta t}(u_h(0)\cos(\omega t) + (\frac{\dot{u_h}(0)}{\omega} - \frac{\beta u_h(0)}{\omega})\sin(\omega t)) \tag{2.21}$$

$\dot{u_h}$ can be set to equal 0 since the system is under consideration at its initial state from rest (i.e. no stored energy). Simplifying and replacing $u_h(0)$ with A:

$$u_h(t) = Ae^{-\beta t}(\cos(\omega t) + (\frac{\beta}{\omega})\sin(\omega t)) \tag{2.22}$$

A final expression for A requires the particular solution and will be found in a later step.

To find an expression for β, an assumption that the system follows a shift dynamic must first be made:

$$u_h(t+1) = \frac{1}{2}u_h(t) \tag{2.23}$$

The rationale for this assumption is introduced in [25]. Substituting with (2.8) yields:

$$e^{-\beta(t+1)}e^{-j2\pi f(t+1)} + e^{-\beta(t+1)}e^{j2\pi f(t+1)} = e^{-\beta t}e^{-j2\pi ft} + e^{-\beta t}e^{j2\pi ft} \tag{2.24}$$

Solving for β:

$$\beta = -\ln\left(\frac{e^{j2\pi f}(1 + e^{j4\pi ft})}{2(1 + e^{j4\pi f(t+1)})}\right) - j2\pi n \tag{2.25}$$

$$\beta = \ln\left(\frac{2(1 + e^{j4\pi f(t+1)})}{e^{j2\pi f}(1 + e^{j4\pi ft})}\right) - j2\pi n \tag{2.26}$$

$$\beta = \ln(2) + \ln\left(\frac{(1+e^{j4\pi f(t+1)})}{e^{j2\pi f}(1+e^{j4\pi ft})}\right) - j2\pi n \tag{2.27}$$

$$\beta = \ln(2) - \ln(e^{j2\pi f}) + \ln\left(\frac{(1+e^{j4\pi f(t+1)})}{(1+e^{j4\pi ft})}\right) - j2\pi n \tag{2.28}$$

To simplify the expression, the third term can be expanded using the property $\ln(\frac{a}{b}) = \ln(a) - \ln(b)$:

$$\beta = \ln(2) - j2\pi f + (1+e^{j4\pi f(t+1)}) - (1+e^{j4\pi ft}) - j2\pi n, \quad n \in \mathbb{Z} \tag{2.29}$$

Because of their periodicity, the third and fourth terms cancel, simplifying to:

$$\beta = \ln(2) - j2\pi f - j2\pi n, \quad n \in \mathbb{Z} \tag{2.30}$$

Taking the real part of Equation 2.30 leads to a final expression for β:

$$\beta = \ln(2) \tag{2.31}$$

Evaluating the general solution u_g can now continue. This will be in the form:

$$u_g(t) = u_h(t) + u_p(t) \tag{2.32}$$

where $u_h(t)$ is the homogeneous solution given by Equation 2.22 and $u_p(t)$ is the particular solution arising from the forcing function. To find $u_p(t)$, an assumption is made that it will be in the same form as the forcing function (the negative sign is added to produce a positive final expression):

$$u_p(t) = -C \tag{2.33}$$

Solving for C, Equation 2.33 is substituted in Equation 2.1:

$$-\ddot{C} - 2\beta\dot{C} - (\omega^2 + \beta^2)C = s(t) \tag{2.34}$$

Because C is a constant, this simplifies to:

$$-(\omega^2 + \beta^2)C = s(t) \tag{2.35}$$

$$C = -\frac{s(t)}{\omega^2 + \beta^2} \tag{2.36}$$

The general solution $u_g(t)$ can be written as:

$$u_g(t) = Ae^{-\beta t}(\cos(\omega t) + (\frac{\beta}{\omega})\sin(\omega t)) - \frac{s(t)}{\omega^2 + \beta^2} \tag{2.37}$$

As in [25], $s(t)$ can be defined as:

$$s(t) = \begin{cases} 1 & 0 \leq t < 1 \\ 0 & t \geq 1 \end{cases} \tag{2.38}$$

A can now be found using the initial conditions $u_g(0) = 0$ and $\dot{u_g}(0) = 0$; these correspond to the initial values of the system starting at $t = 0$ with no stored energy. Substituting:

$$0 = Ae^{-\beta * 0}(\cos(\omega * 0) + (\frac{\beta}{\omega})\sin(\omega * 0)) - \frac{s(t)}{\omega^2 + \beta^2} \tag{2.39}$$

$$0 = A - \frac{s(t)}{\omega^2 + \beta^2} \tag{2.40}$$

$$A = \frac{s(t)}{\omega^2 + \beta^2} \tag{2.41}$$

The value for $s(t)$ can be found by evaluating Equation 2.38 at $t = 0$, which results in:

$$A = \frac{1}{\omega^2 + \beta^2} \tag{2.42}$$

Finally, the complete general solution can be written:

$$u_g(t) = \frac{1}{\omega^2 + \beta^2}\left[1 - e^{-\beta t}(\cos(\omega t) + (\frac{\beta}{\omega})\sin(\omega t)) - \frac{s(t)}{\omega^2 + \beta^2}\right] \tag{2.43}$$

Splitting into cases based on the value of $s(t)$ and adding an expression for the system at rest when $t < 0$:

$$u_g(t) = \begin{cases} 0 & t < 0 \\ \frac{1}{\omega^2+\beta^2}\left[1 - e^{-\beta t}(\cos(\omega t) + (\frac{\beta}{\omega})\sin(\omega t))\right] & 0 \leq t < 1 \\ \frac{1}{\omega^2+\beta^2}\left[e^{-\beta t}(\cos(\omega t) + (\frac{\beta}{\omega})\sin(\omega t))\right] & t \geq 1 \end{cases} \tag{2.44}$$

Equation 2.44 forms a basis function for the reverse time system in a manner congruous with the forward time system. A plot of this function evaluated for $0 \leq t < 3$ is shown in the following figure.

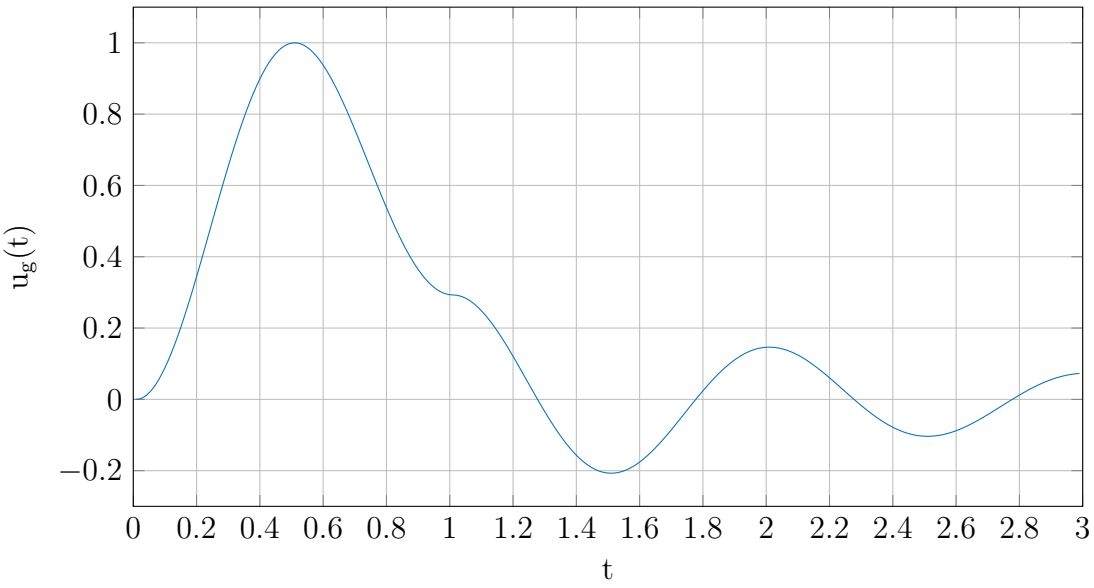

Figure 2.4: Reverse Time Basis Pulse

The difference between the two cases is clearly visible; for $0 \leq t < 1$, the basis pulse is offset by 1 while for $t \geq 1$, this offset is removed. As expected, this response is identical

but reversed in time to the forward time basis pulse in Equation 1.19. Because the system's solution can be described in terms of this basis pulse, it can be extended to all t as given in [25]:

$$u(t) = \sum_{n=-\infty}^{t} s_n u_g(t-n) \tag{2.45}$$

The existence of this solution also allows for the development of a matched filter with the same process used in [4]. A detailed development of the filter is presented in the next chapter.

2.3 Chaotic Maps

While the expression in Equation 1.13 provides a complete description the dynamics of the chaotic system for any time point, it is not necessary use its complete solution for many types of analysis. A common method to explore the behavior of chaotic systems is to find a one-dimensional map that describes the system at some fixed time interval in terms of its state one time step earlier [35]. Mathematically, this can be described as:

$$x(t+1) = Fx(t) \tag{2.46}$$

where F is a function that established the mapping. Many types of chaotic behavior can be described effectively with the appropriate definition of F [36, 37].

2.3.1 Shift Map

A chaotic shift map describing Equation 1.13 operating in the shift band can be defined as:

$$u_{n+1} = e^{\beta} u_n - (e^{\beta} - 1) s_n \tag{2.47}$$

$$s_n = \begin{cases} +1, & u_n \geq 0 \\ -1, & u_n < 0 \end{cases} \tag{2.48}$$

where the map is piecewise linear and bounded such that $|u_n| \leq 1$ and $0 < \beta \leq \ln(2)$ [4]. This map is shown graphically in the following figure.

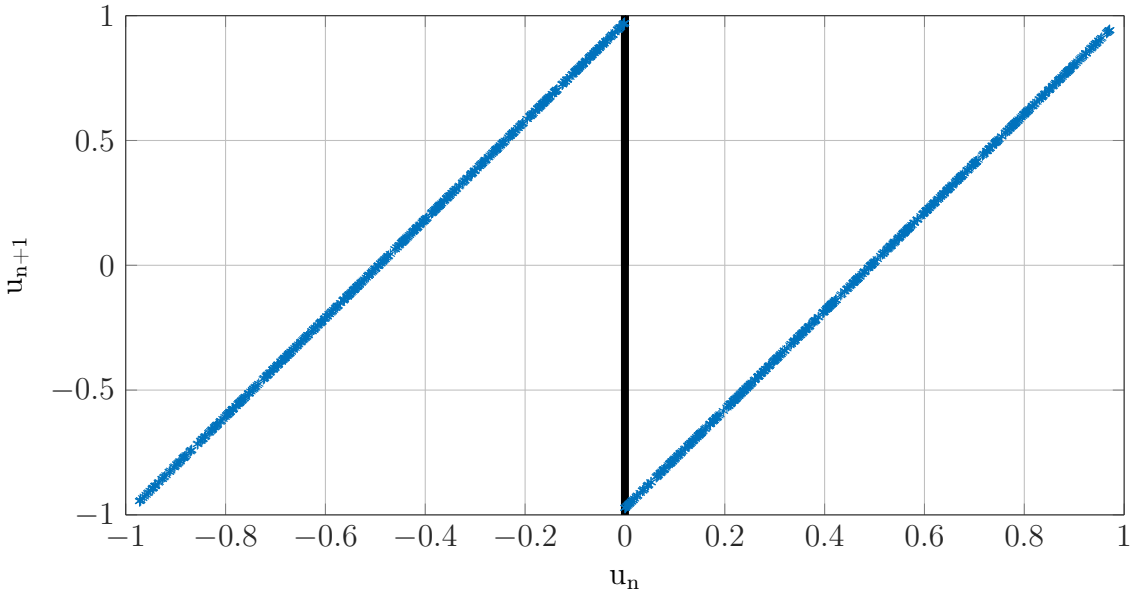

Figure 2.5: Return Map of Shift-band System

While Figure 2.5 was generated by iterating through Equation 2.48 many times, it is important to note that this map's values also correspond to return points of Equation 1.13. Each value of the map represents the continuous time value after a single orbit in phase space. Figure 2.6 demonstrates these points.

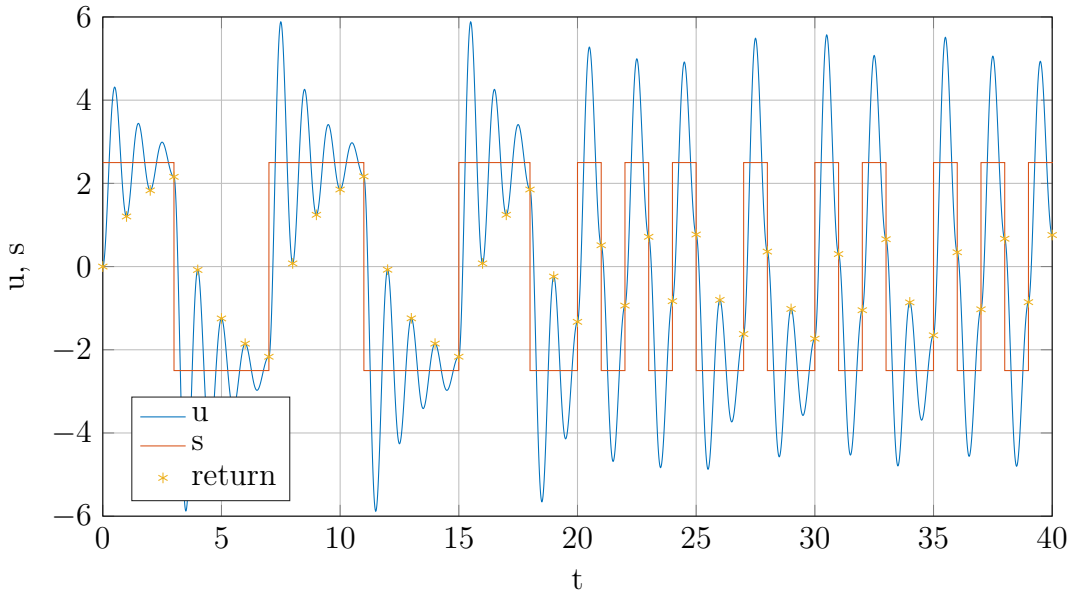

Figure 2.6: Numerical Simulation of Reverse Time System in Shift Band with Return Points (time)

In this figure, each marked return point represents a new value of the map defined by Equation 2.47 - plotting these points would exactly regenerate Figure 2.5 above.

To determine if this map is chaotic, the slope in these bounds can be examined; it evaluates to a constant, e^{β}, which can be shown to always be greater than 1 for any $\beta > 0$. This result proves that the map is chaotic due to its possession of positive entropy.

By varying the initial conditions and β value of Equation 2.47, the symbolic dynamics - which future states the system can take as a function of its current state - of the system can be explored [38, 39]. MATLAB code has been developed that iterates through the map for its full range of possible initial conditions at a given β value; this code is included in the appendices. Using this code in a nested sweep, β was also varied through its allowable values. At each initial condition and β combination, the map was iterated four times so that four s_n values (referred to as symbols) were generated. The result of these sweeps is shown in Figure 2.7.

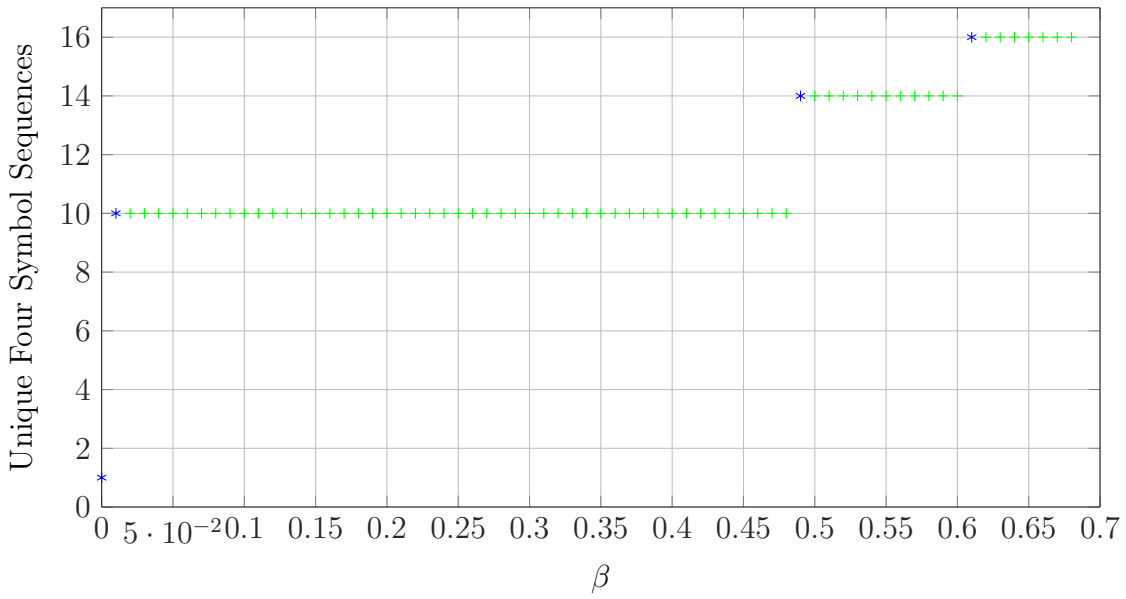

Figure 2.7: Symbolic Dynamics of Shift-band System

For β near $\ln(2)$, any possible combination of s_n values can be generated. Because s_n can take two values, this leads to $2^4 = 16$ possible sequences. As β decreases, the number of possible sequences is reduced, resulting in a restricted grammar.

Examining the relationship between initial conditions and generated symbol sequences for a single β value generates a coding function that maps the two. To allow for plotting, the symbol sequences were converted into decimal; the coding function generated for $\beta = \ln(2)$ is shown in the following figure.

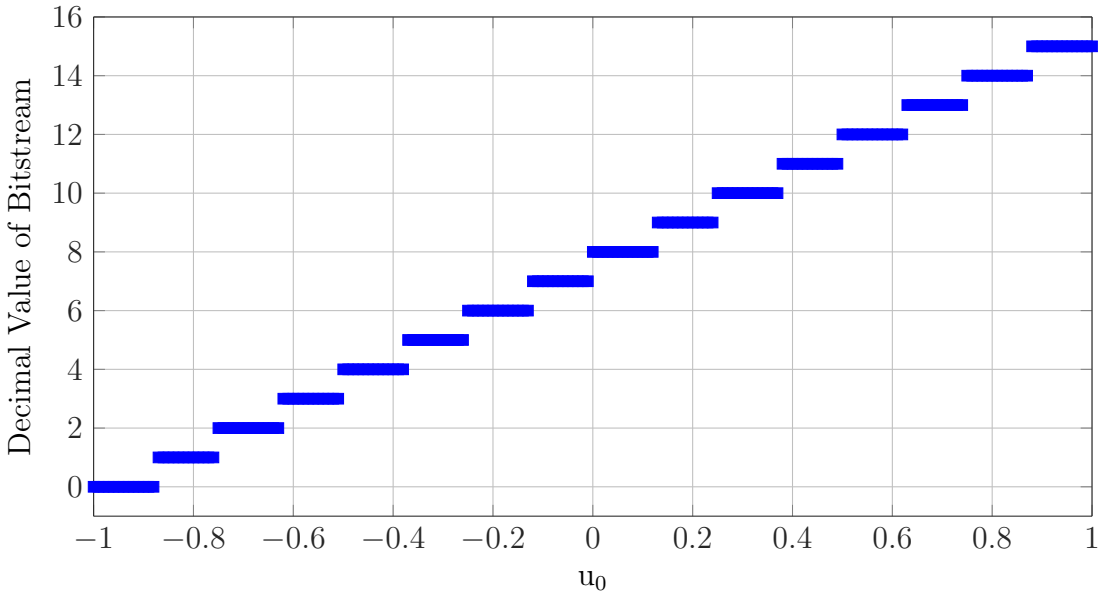

Figure 2.8: Coding Function of Shift-band System for $\beta = \ln(2)$

This β value represents the special case of unrestricted grammar, which results in a coding function that covers both the full range of initial conditions and the entire set of possible symbol sequences (each possible initial condition maps to a symbol sequence). As β is reduced, the effects of grammar restriction become more apparent. Figure 2.9 shows this occurring.

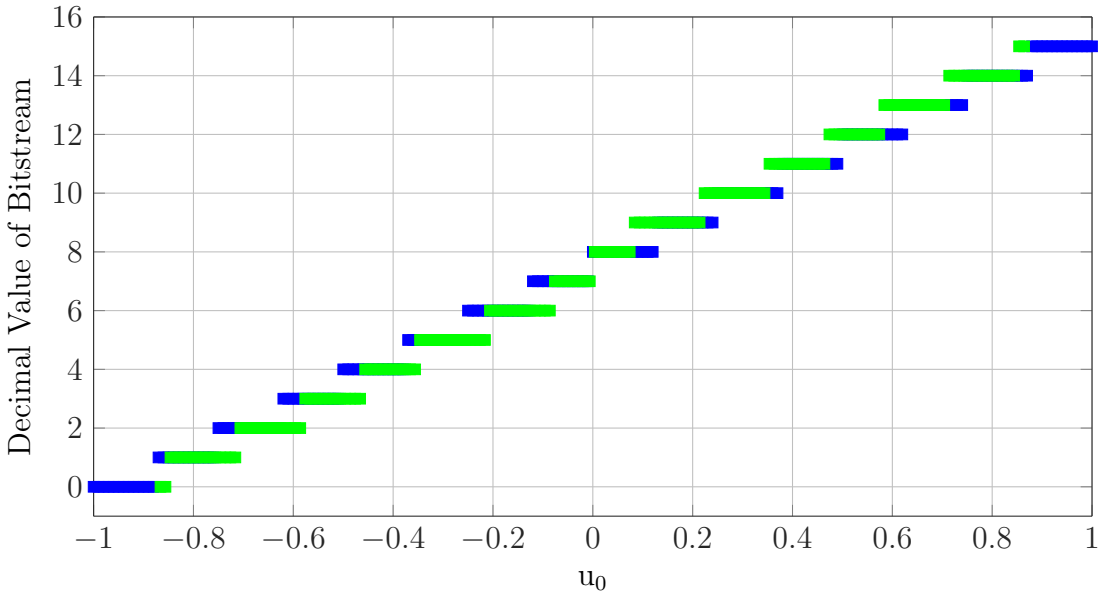

Figure 2.9: Coding Function of Shift-band System for Decreasing β

The value of β added to this figure results in the same set of allowable symbol sequences, but the range of initial conditions that map to a given sequence is reduced. This behavior is visible as the shorter green segments (representing $\beta = 0.9\ln(2)$) overlaid on the blue segments (representing $\beta = \ln(2)$). Functionally, this restricts the system grammar by removing the initial conditions for which there is no green segment from the set of allowable initial conditions for this β value.

If this process is continued further such that β falls to a lower step on Figure 2.7, some symbol sequences are lost entirely. This is shown in Figure 2.10.

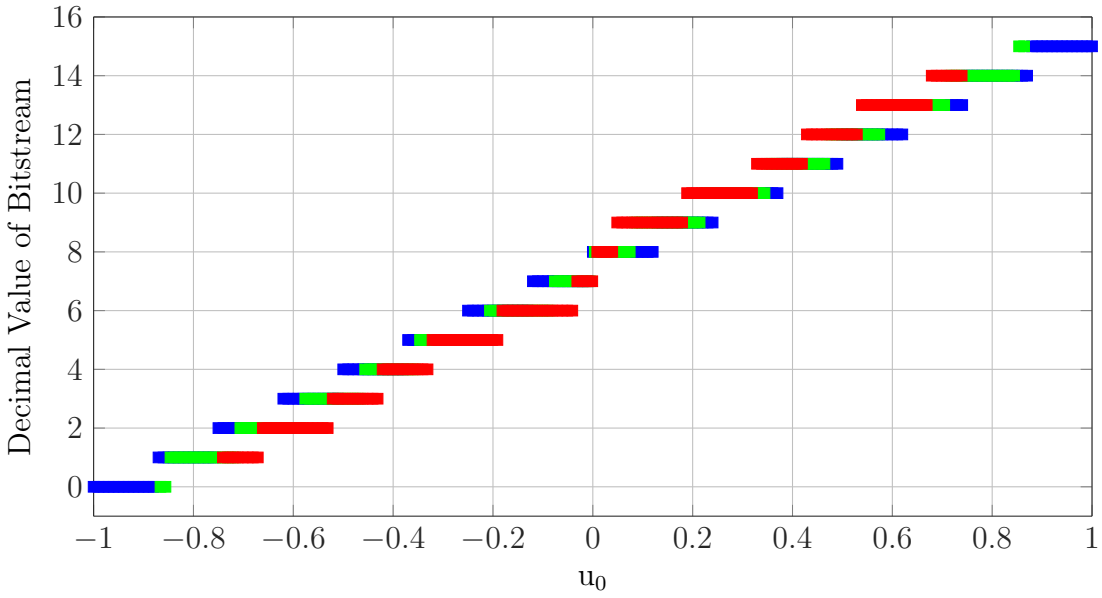

Figure 2.10: Coding Function of Shift-band System for Decreasing β

Building on the previous figure, a red overlay (representing $\beta = 0.8\ln(2)$ is added. In addition to the red segments shrinking further as compared to the blue and green, the two levels at 0 and 16 do not contain a red overlay at all. These levels represent the lost symbol sequences referred to above.

2.3.2 Skew Tent Map

A second chaotic map, the skew tent map, which describes Equation 1.13 operating in the folded-band, is defined in [5] as:

$$u_{n+1} = \begin{cases} e^{\beta}u_n, & 0 < u_n \leq 1 \\ -e^{3\beta/2}u_n + (e^{\beta} + e^{3\beta/2}), & 1 < u_n < 1 + e^{-\beta/2} \\ e^{\beta}u_n - (e^{\beta/2} + e^{\beta}), & u_n > 1 + e^{-\beta/2} \end{cases} \tag{2.49}$$

As with the shift map, a determination on the chaotic nature of this map can be made by examining the slope of the map within its bounds. Although this is not a constant value, it

can be shown that it always possesses a magnitude greater than 1; chaotic behavior is again proven by the positive entropy resulting from any positive value of β.

Unlike the shift map, the skew tent map does not directly yield a s_n sequence. To overcome this limitation, the relation:

$$s_n = \begin{cases} A, & 0 \leq u_n < 1 \\ B, & 1 \leq u_n < 1 + e^{-\beta/2} \\ C, & u_n \geq 1 + e^{-\beta/2} \end{cases} \tag{2.50}$$

as defined in [5], where the symbols (which correspond to the three partitions of the map) are mapped as $A \rightarrow 00$, $B \rightarrow 100$, and $C \rightarrow 10$, can be used. Figure 2.11 shows these partitions graphically.

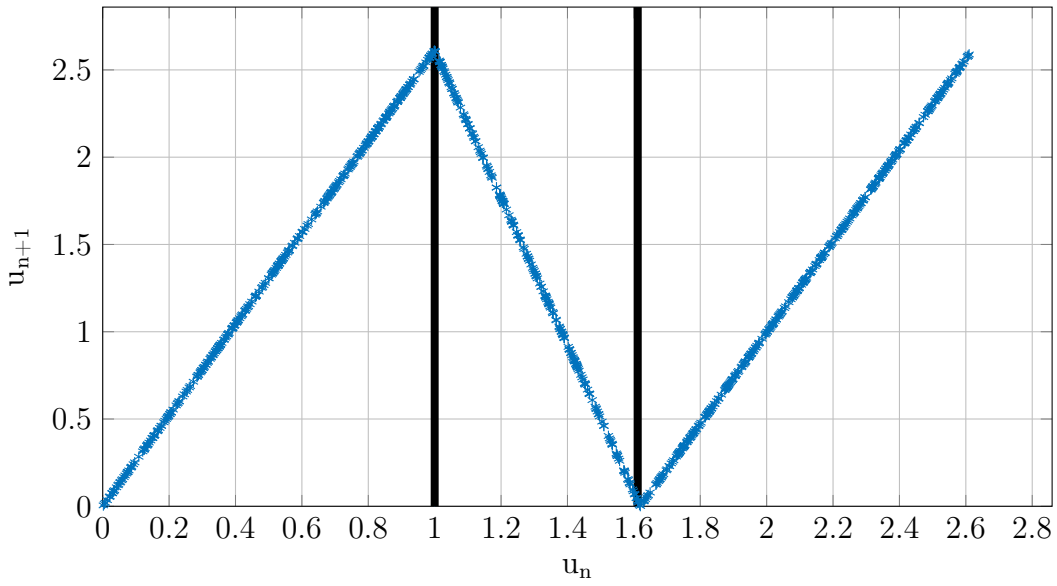

Figure 2.11: Return Map of Folded-band System

To account for the A, B, and C substitutions being unequal in length, a relation for the time between each s_n value can be defined as:

$$t_{n+1} = \begin{cases} t_n + 1, & 0 < u_n \leq 1 \\ t_n + \frac{3}{2}, & 1 < u_n < 1 + e^{-\beta/2} \\ t_n + 1, & u_n > 1 + e^{-\beta/2} \end{cases} \tag{2.51}$$

This variable timing arises from the need to account for the extra half cycle that occurs when the systems enters its fold (see Figure 1.15).

As with the shift map, the tent map can be iterated while varying its initial conditions and β to provide an understanding of the system's symbolic dynamics. Because the tent map's s_n consists of multi-bit symbols, the maximum number of sequences is found by considering the longest symbol, $B \rightarrow 100$, which results in $3^4 = 81$. The results from these sweeps are shown in the following figure.

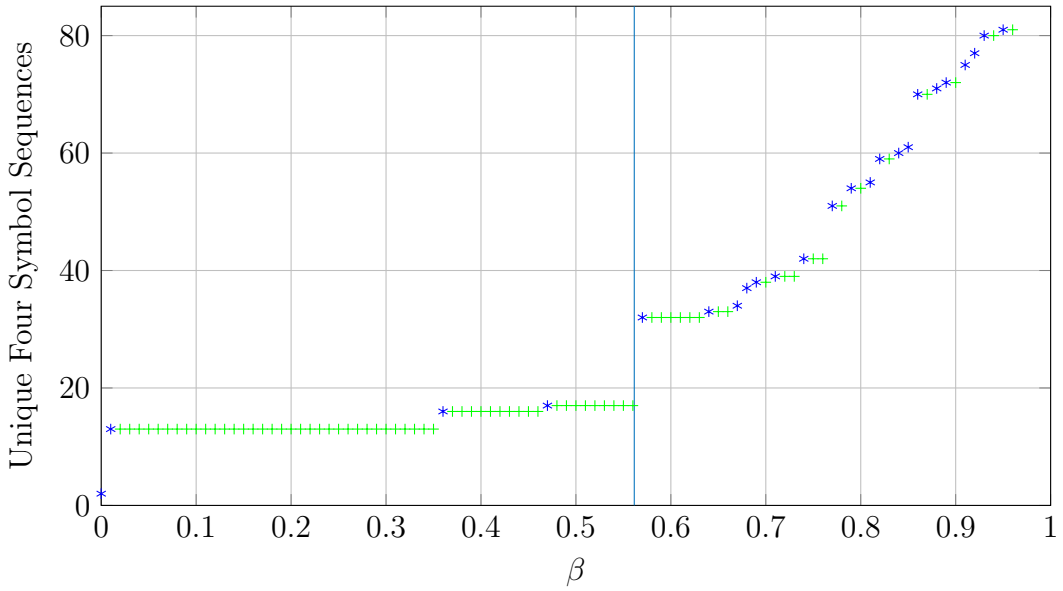

Figure 2.12: Symbolic Dynamics of Folded-band System

The line marking $0.81 * \ln(2)$ indicates the β value below which the third symbol C disappears entirely. For the values above this line, the number of symbol sequences changes quickly as β is decreased; below this line, the behavior is analogous to the shift map.

Coding functions were also generated for different β values showing the effects of moving from unrestricted grammar to an increasingly restricted grammar. These are shown in the following three figures and follow the same order of decreasing value and color as the shift map coding function figures.

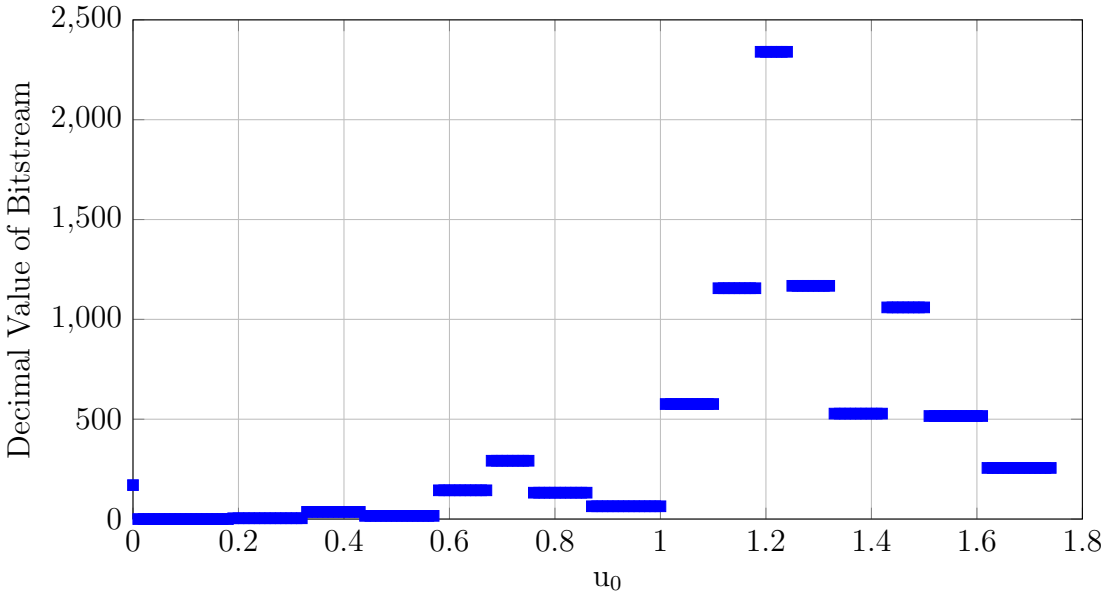

Figure 2.13: Coding Function of Folded-band System for $\beta = 0.81\ln(2)$

For $\beta = 0.81\ln(2)$, which is the maximum value for generating only symbols A and B, each initial condition maps to an output symbol sequence. This mapping differs significantly from those shown in Figure 2.8, confirming that the skew tent map possesses significantly different dynamics when compared with the shift map. As β is reduced, the effects of grammar restriction become visible. Figures 2.14 and 2.15 show this occuring for $\beta = 0.71\ln(2)$ (green) and $\beta = 0.61\ln(2)$ (red), respectively.

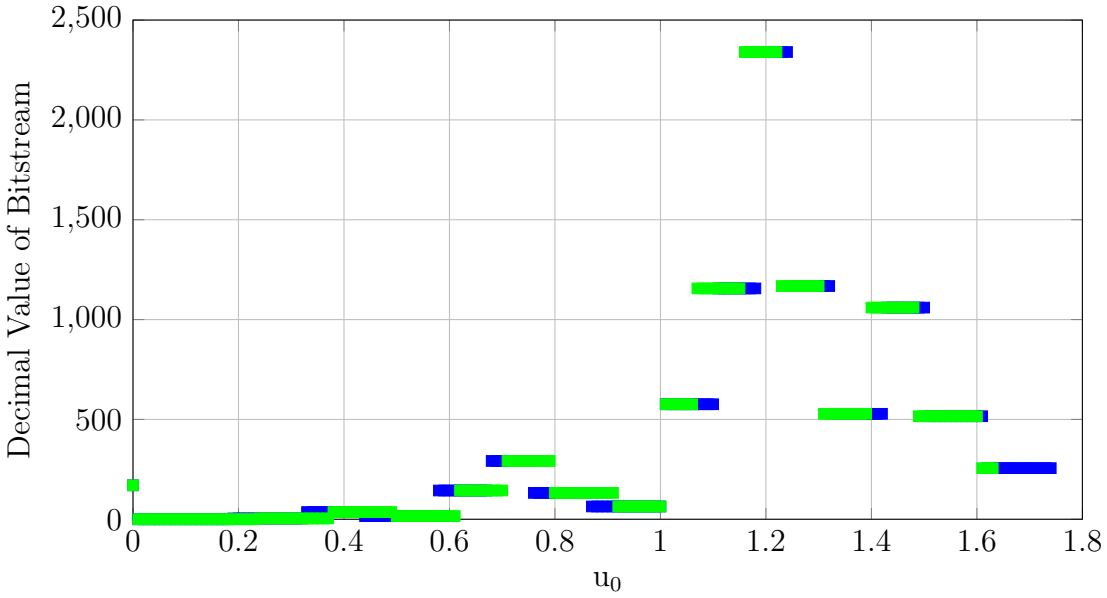

Figure 2.14: Coding Function of Folded-band System for Decreasing β

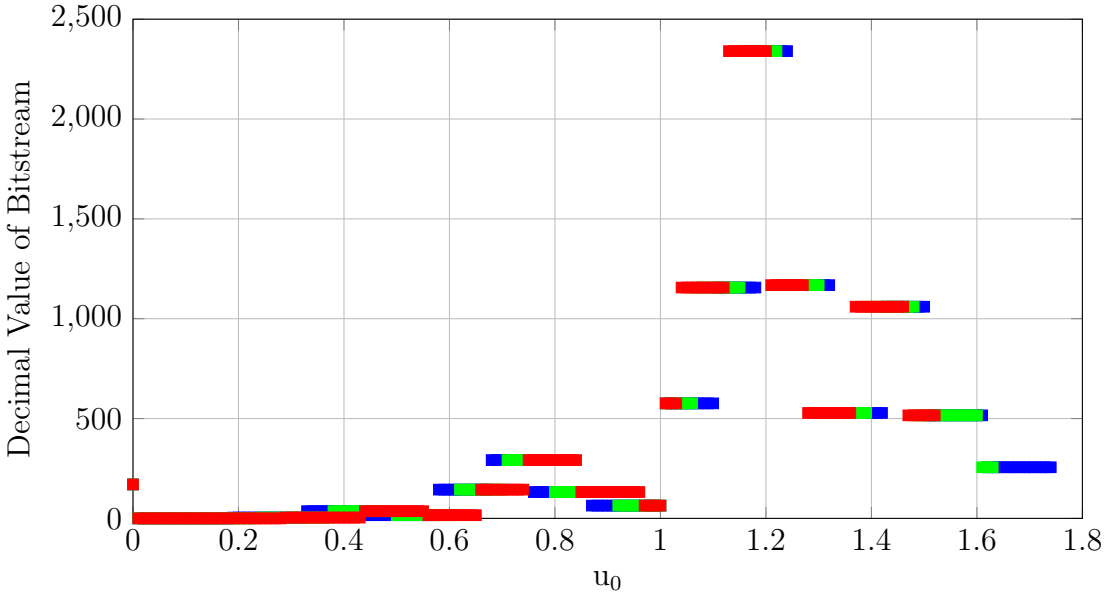

Figure 2.15: Coding Function of Shift-band System for Decreasing β

2.3.3 Sequence Generation

Both the shift map and the skew tent map have been shown to generate chaotic sequences when given appropriate initial conditions and control parameters. A summary of these is provided in Table 2.1.

Map	β
shift	$0 < \beta < \ln(2)$
skew tent (2 partitions)	$0 < \beta < 0.81ln(2)$
skew tent (3 partitions)	$0 < \beta < 1.39ln(2)$

Table 2.1: Summary of Chaotic Map Parameters

Previous work has shown that a random sequence for s_n in Equation 2.2 can be replaced by sequences generated from chaotic iterated maps to realize different types of dynamics in the reverse time system. Both Lorenz like and Rössler like dynamics have been observed in numerical calculations when driving the system with random sequences using different encodings [40]. Building on these conclusions, the s_n sequences generated from the chaotic maps presented above can be used to generate these encoded random sequences [41]. By selecting desired initial condition and β combinations, precise control over the reverse time chaotic dynamics is possible.

2.4 Circuit Model

In order to implement reverse time chaos in hardware, components realizing both the continuous time state u and the s_n drive sequence have been developed. To aid in this development, the system described in equation (2.1) is rewritten in terms of electrical circuit components as:

$$\frac{\mathrm{d}^2 V_{RT}}{\mathrm{d}\tau^2} + \frac{TR}{L}\frac{\mathrm{d}V_{RT}}{\mathrm{d}\tau} + \frac{T^2}{LC}V_{RT} = f(\tau) \tag{2.52}$$

$$f(\tau) = \frac{T^2 V_{in}}{LC} s_n, \; n \leq \tau < n+1 \tag{2.53}$$

where the time constant τ is defined as $\tau = \frac{t}{T}$ and V_{in} is an arbitrary fixed magnitude [25]. A diagram illustrating a circuit implementation of this system is shown in the Fig. 2.16.

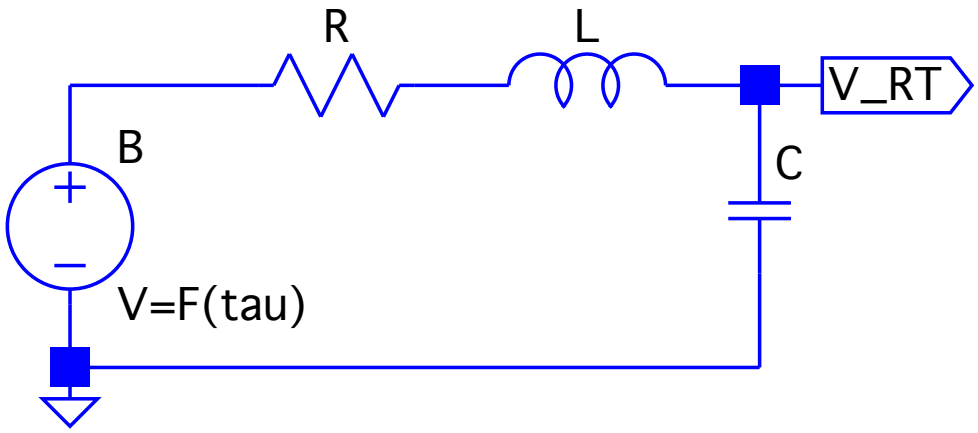

Figure 2.16: Reverse Time System Circuit Model

The values for the R, L, and C components can be determined by solving the system of equations given by:

$$\frac{TR}{L} = 2\ln(2) \tag{2.54}$$

$$\frac{T^2}{LC} = 4\pi^2 + \ln(2)^2 \tag{2.55}$$

where one of the component values must be set arbitrarily so that the two equations contain only two unknowns.

To generate the $f(\tau)$ signal, s_n can be generated computing the map given in Equations 2.47 and 2.48 (for shift map dynamics) or Equations 2.49 and 2.50 (for tent map dynamics). Any allowed value of β can be used with this method, which allows for operation with arbitrary grammar restrictions (or with unrestricted grammar). Control of the reverse time system can be realized by starting the map computation with the initial condition (u_0) that corresponds to the desired s_n sequence.

2.4.1 HDL

The s_n sequence was generated by implementing the shift map described above in VHDL. A complete listing of the code used for both bands is provided in the appendices. Signed Qm.n fixed-point numerical representations were used to allow for high speed operation. In this scheme, numbers are represented in binary where Qm.n maps to 1 sign bit, m-1 integer bits, and n fractional bits and results in a range of $[(-2^m).(\frac{1}{2^n}), (2^m - 1).(\frac{1}{2^n})]$. High accuracy was maintained by using 32 bit variables in Q4.28 format, which allowed for an integer range of -8 to 7 and a minimum resolution of $\frac{1}{2^{28}}$ (3.725e-9).

Map calculations were split into multiple clock cycles in order allow for synthesis into physical circuits. This was required due to the expressions in both maps containing multiple mathematical operations; in hardware, one operation must complete before the next can use its value. Due to this, a new map value is generated every two cycles of the input clock for the shift map and every two (partitions A and C) or three (partition B) cycles of the input clock for the skew tent map. Mentor Graphics ModelSim was used to simulate the VHDL model and produce output bitstreams used in a complete simulation.

2.4.2 SPICE

A SPICE simulation of the hardware described in the previous section was constructed initially to predict how an actual circuit would perform at a target operating frequency of 1.8 MHz. LTSpiceIV from Linear Technology [42] was used to perform the simulations. Component values for this frequency were calculated to be R=250 Ω (fixed), L=100 μH, and C=75 pF using Equations 2.54 and 2.55, and the HDL simulation output was used as the input voltage representing $f(\tau)$. These SPICE results are shown in Figure 2.17 for the shift map with $\beta = \ln(2)$.

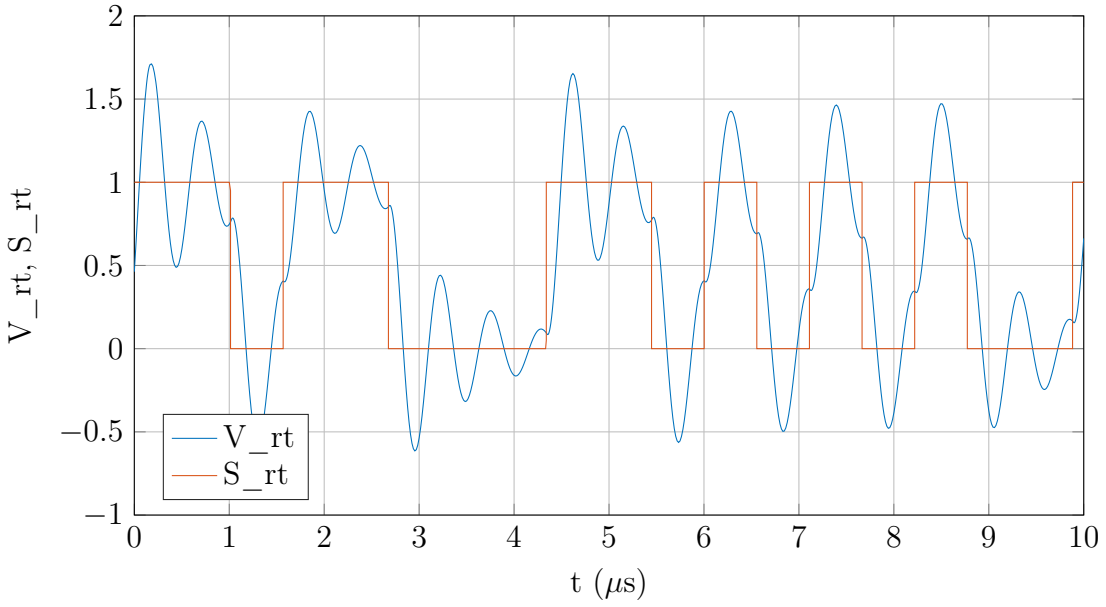

Figure 2.17: Simulation Results of Shift-band System (time)

The simulation results produce the expected chaotic behavior. It should be noted that the simulated S_rt was left at its default output range of 0-1 V; although this differs from the s_n definition given in Equation 2.48, the shift is accounted for by the V_{in} term in Equation 2.53 and the resulting system dynamics mimic the original system's behavior around these new levels.

A complete simulation model is included on the following page. Buffers were added to both sides of the RLC tank and a level shifting resistor divider was added to the output to allow for interfacing with other circuitry. The C value `C3` was adjusted slightly from 75 pF to 68 pF to match with on hand parts. As shown, the opamp selected for use in the buffers was the Linear Technology LT1220; this selection was made due to both high performance and availability in a readily solderable DIP package [43].

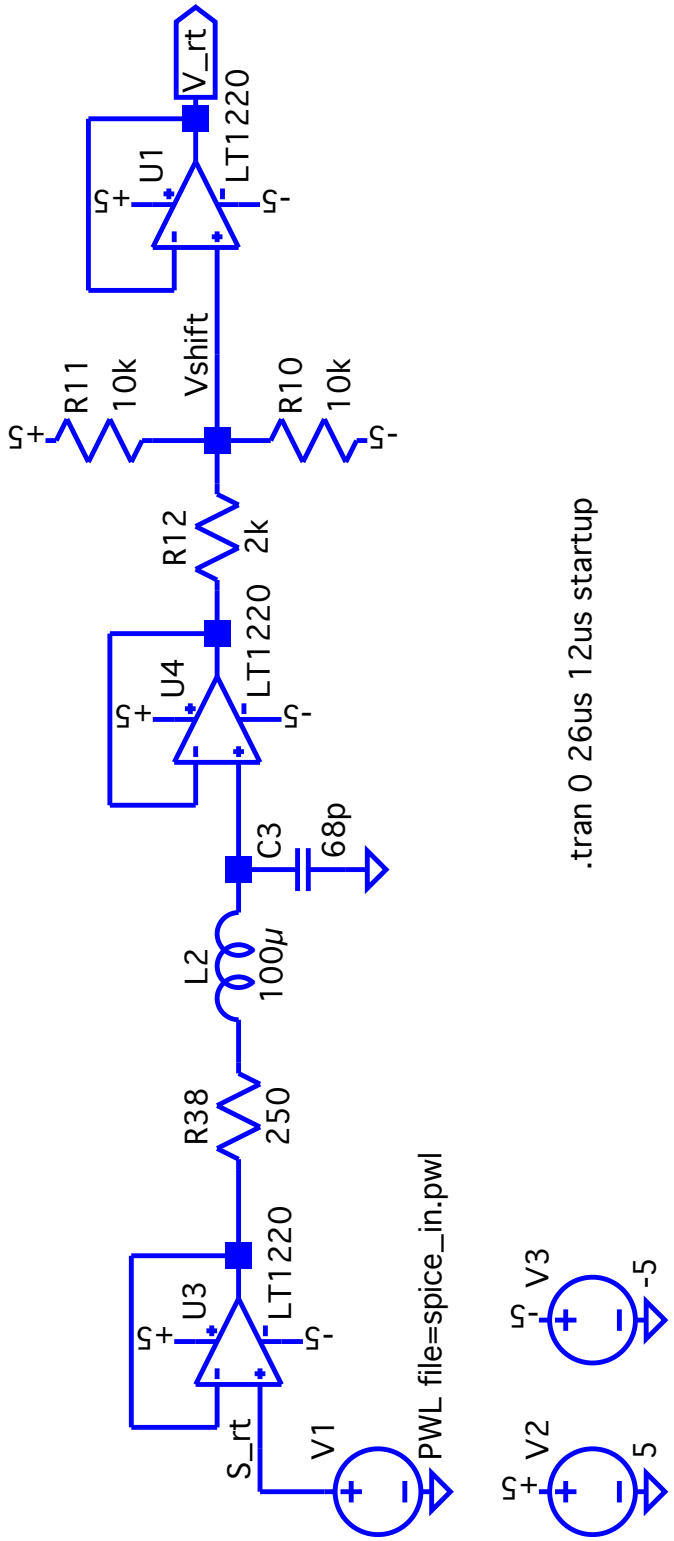

Figure 2.18: Reverse Time System Simulation Model

2.5 Hardware

2.5.1 Components

To realize the model used for simulation in hardware, a PCB was designed and assembled. The circuit used for the PCB layout closely follows the schematic shown in Figure 2.18 with the addition of decoupling capacitors to remove noise from the power supply lines and the replacement of the resistor divider with a summing amplifier tunable with the potentiometer `RV101`. The assembled board is shown in Figure 2.19.

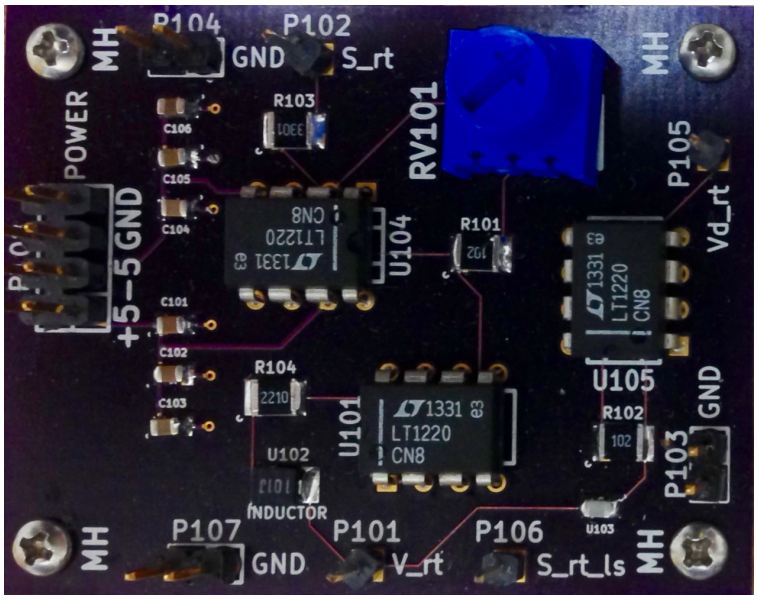

Figure 2.19: Reverse Time Oscillator PCB

Testing of the reverse time oscillator was performed by using a low-cost Altera Cyclone IV-E FPGA clocked at 3.6 MHz (twice the tuned operating frequency of the RLC tank) to generate a s_n sequence that was fed into the oscillator PCB as $f(\tau)$. An image of this setup is presented in the following figure.

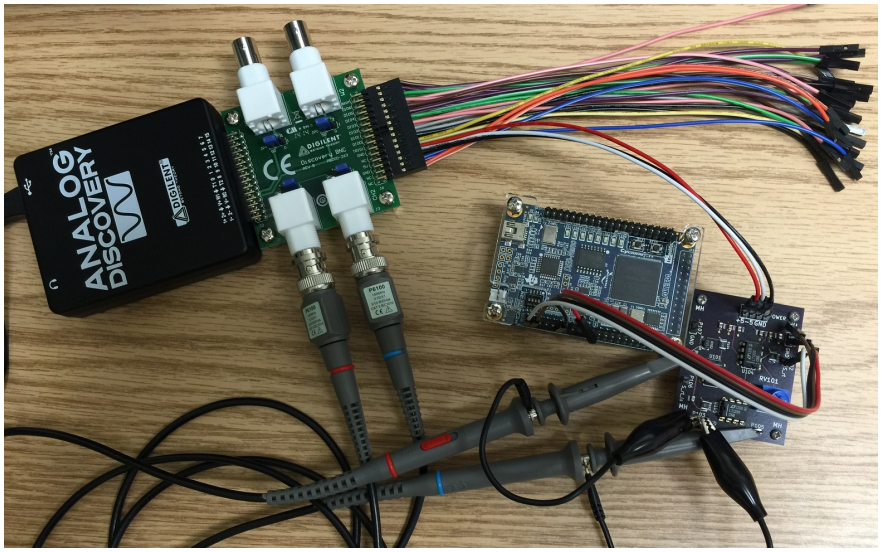

Figure 2.20: Reverse Time Oscillator Testing Setup

2.5.2 Measurement Results

Measurements were taking using the Digilent Analog Discover USB oscilloscope. The analog bandwidth was set to 10 MHz to allow for complete capture of the chaotic waveforms as well as faster transients resulting from the switching of $f(\tau)$. Regulated power rails (+5 V and -5 V) and a ground reference were also provided by the Analog Discovery board. Screen captures are used for each measurement to show the exact settings used for capture. Results from operation using the shift map with $\beta = \ln(2)$ to generate $f(\tau)$ are shown in Figure 2.21.

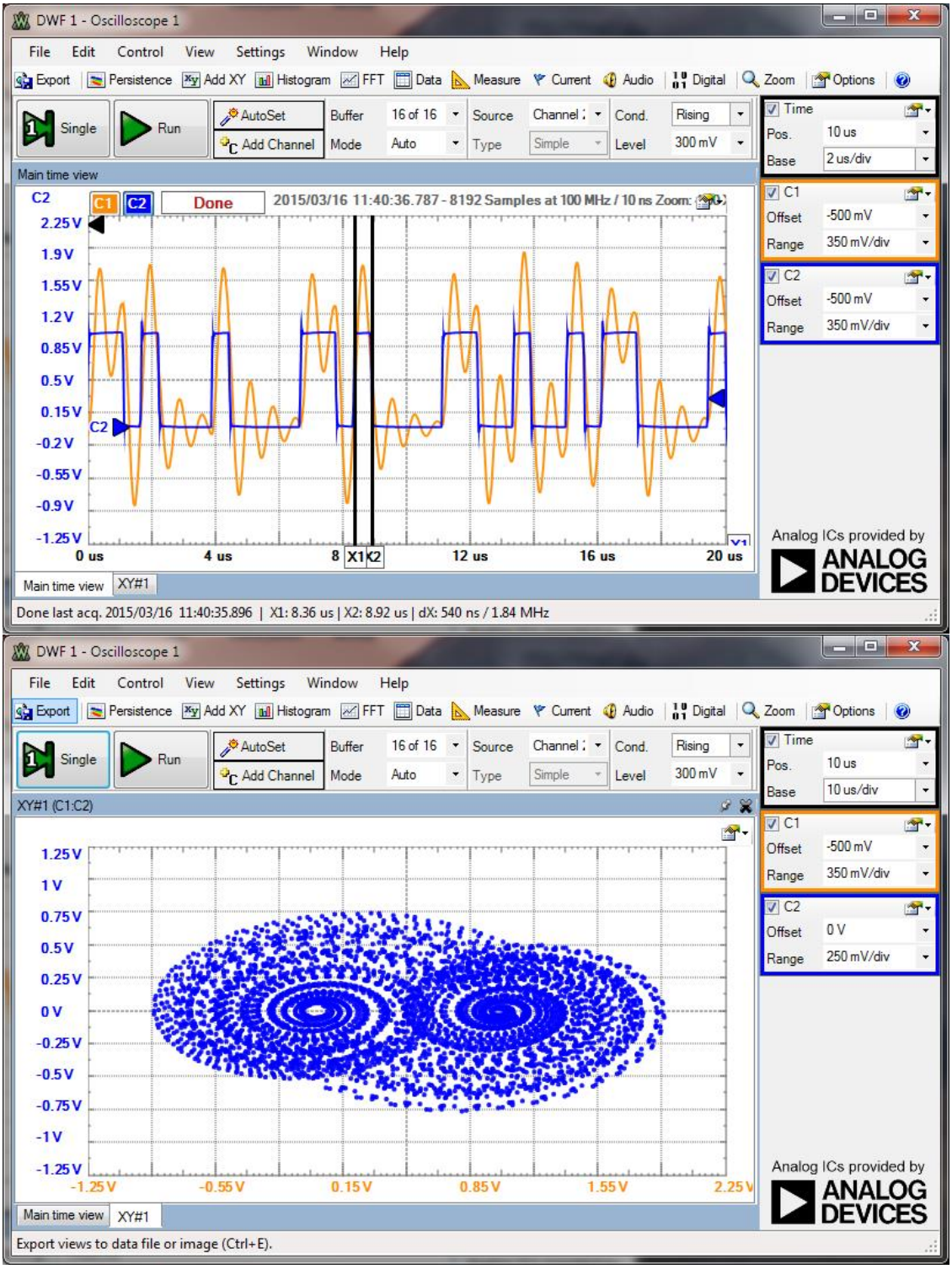

Figure 2.21: Measurement Results using Shift Map (top: time, bottom: phase portrait)

As expected, the hardware produces reverse time dynamics which are similar to those of the system in [4] and are in close agreement to the simulation results shown in Figure 2.17. The double-scroll attractor shown in the phase portrait clearly indicates chaotic behavior.

The real time FFT function of the oscilloscope also allowed for exploration of the spectral content of the chaotic waveform. A flat response from DC to approximately the tuned frequency of the RLC tank confirms spread spectrum operation; the sharp dip near 1.8 MHz results from the impedance of the tank falling close to zero at resonance. This output is shown in Figure 2.22.

Figure 2.22: Measurement Results using Shift Map (spectrum)

Similarly, when operated using the tent map with $\beta = 1.39\ln(2)$, the hardware's dynamics are a reverse time analogue to those of the system presented in [5]. Chaotic behavior is again confirmed by examining the phase space. Figure 2.23 provides these results.

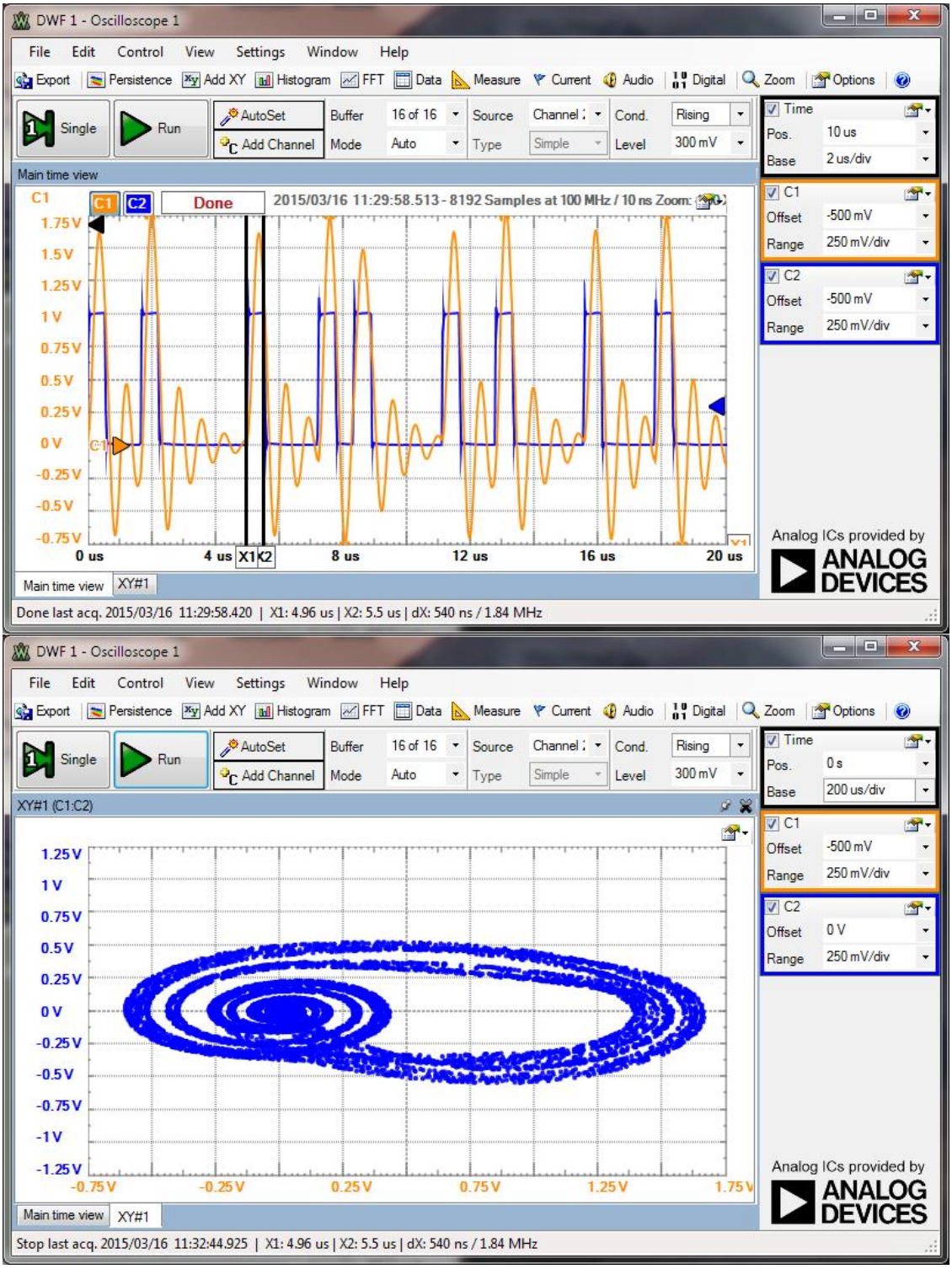

Figure 2.23: Measurement Results using Tent Map (top: time, bottom: phase portrait)

The stretched appearance of the fold in the phase portrait is due to the x axis scale used for plotting. Closer observation confirms that the center of the fold is located at approximately 1, which preserves the Rössler like dynamics. As when operating with the shift map, the spectrum of the reverse time system operating with the tent map was observed to be approximately flat approaching the resonant frequency of the tank. This is shown in the following figure.

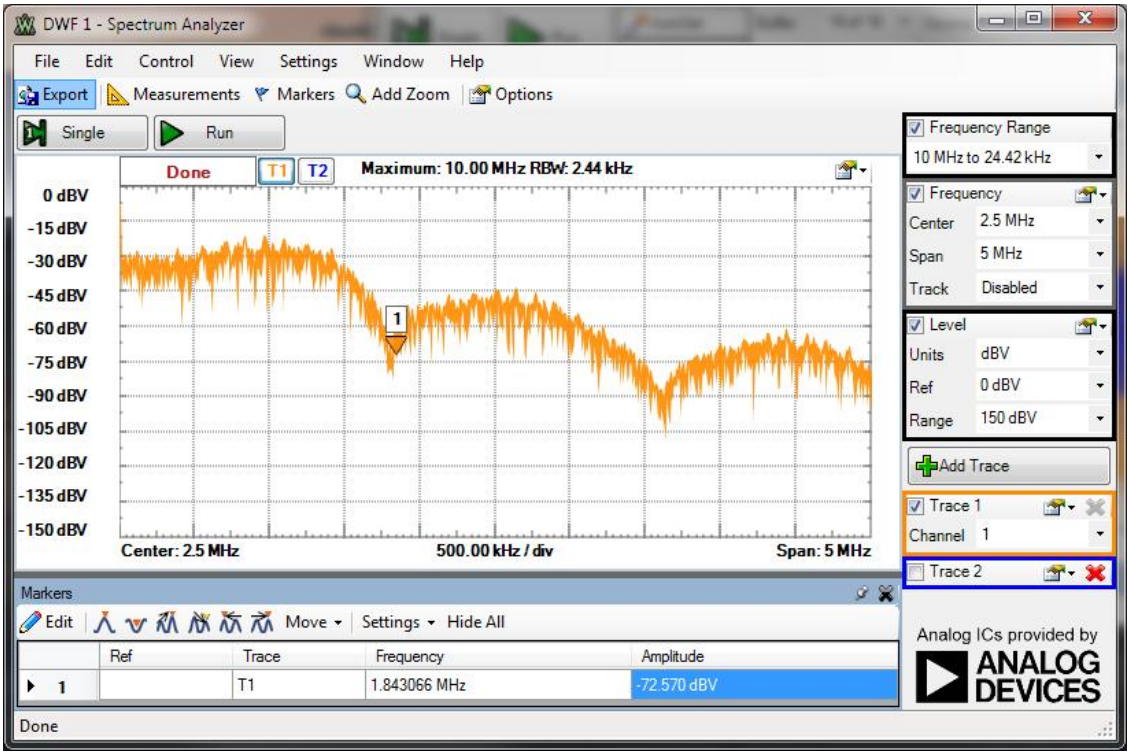

Figure 2.24: Measurement Results with Skew Tent Map (spectrum)

Chapter 3

Digital Matched Filter

3.1 Theory

A common problem in many signal processing applications is to determine if a known signal is present in a received signal which may contain significant noise. Although there are multiple approaches which solve this problem, the use of a filter whose response is matched to the known signal - a matched filter - is frequently the solution of choice. This matched filter can be described as the system with the maximal output SNR when presented with a known signal corrupted by AWGN . A mathematical derivation, based on the coverage in [44], [45], and [46], follows.

3.1.1 Derivation

Given a known transmitted signal $x(t)$, the received signal $v(t)$ can be written as:

$$v(t) = Ax(t - t_p) + n(t) \tag{3.1}$$

where A is amplitude scaling due to attenuation or gain of the channel, t_p is the propagation delay through the channel, and $n(t)$ is AWGN added in the channel with a PSD $S_n(\omega)$. The SNR of this system can be defined as:

$$\text{SNR} = \frac{\text{peak signal power}}{\text{average noise power}} = \frac{\max\left(|Ax(t - t_0)|^2\right)}{\overline{|n(t)^2|}} = \frac{|Ax(0)|^2}{\overline{|n(t)|^2}} \tag{3.2}$$

The matched filter is the linear filter ($h(t)$ or $H(\omega)$) that maximizes its output SNR at some instant relative to a starting point t_0. The output from such a filter can be described

by:

$$v_o(t) = v(t) * h(t) = y(t) + n(t) \tag{3.3}$$

To achieve the maximal SNR, an $h(t)$ must be found such that:

$$\text{SNR} = \frac{|y(t_x)|^2}{\overline{|n_o(t)|^2}} \tag{3.4}$$

at some instant $t_x > t_o$. The input to the matched filter is defined as:

$$v(t) = x(t) + n(t) \tag{3.5}$$

The output of the matched filter is expanded from Equation 3.3 to:

$$v_o(t) = x(t) * h(t) + n(t) * h(t) \tag{3.6}$$

where $y(t) = x(t) * h(t)$ is the output signal component and $n_o(t) = n(t) * h(t)$ is the output noise component. Recall Equation 3.4:

$$\text{SNR} = \frac{|y(t_x)|^2}{\overline{|n_o(t)|^2}} = \frac{\text{peak signal power at } t = t_x}{\text{average noise power}} \tag{3.7}$$

An expression for $y(t)$ in the frequency domain can be written as:

$$y(t) = x(t) * h(t) = \frac{1}{2\pi} \int X(\omega) H(\omega) e^{j\omega t} \mathrm{d}\omega \tag{3.8}$$

With this, the peak signal power at $t = t_x$, $|y(t_x)|^2$, can be found:

$$|y(t_x)|^2 = \left| \frac{1}{2\pi} \int X(\omega) H(\omega) e^{j\omega t_x} \mathrm{d}\omega \right|^2 \tag{3.9}$$

A similar methodology can be used to find the average noise power $\overline{|n_o(t)|^2}$ and results in:

$$\overline{|n_o(t)|^2} = \frac{1}{2\pi}\int S_{n_o}(\omega)\mathrm{d}\omega = \frac{1}{2\pi}\int S_n(\omega)|H(\omega)|^2\mathrm{d}\omega \tag{3.10}$$

Equation 3.10 can be simplified by taking advantage of the fact that $S_n(\omega)$ describes AWGN to rewrite it as a constant term $\frac{N_0}{2}$:

$$\overline{|n_o(t)|^2} = \frac{N_0}{4\pi}\int |H(\omega)|^2\mathrm{d}\omega \tag{3.11}$$

Next, an expression for $H(\omega)$ must be developed for maximal SNR at $t = t_x$. This can be accomplished using the results from above:

$$\frac{|y(t_x)|^2}{\overline{|n_o(t)|^2}} = \frac{\left|\frac{1}{2\pi}\int X(\omega)H(\omega)e^{j\omega t_x}\mathrm{d}\omega\right|^2}{\frac{N_0}{4\pi}\int |H(\omega)|^2\mathrm{d}\omega} \tag{3.12}$$

Equation 3.12 does not readily lend itself to analysis. To simplify, the Schwartz inequality can be used. This states that, for two complex functions $f(x)$ and $g(x)$, integrable over $[a, b]$:

$$\left|\int_a^b f(x)g(x)\mathrm{d}x\right|^2 \leq \left|\int_a^b f(x)\mathrm{d}x\right|^2 \left|\int_a^b g(x)\mathrm{d}x\right|^2 \tag{3.13}$$

Notably, the LHS and RHS are equal for the special case where:

$$g(x) = kf^*(x), k \in \mathbb{R} \tag{3.14}$$

To apply the Schwartz inequality to the numerator of Equation 3.12, $f(x)$ and $g(x)$ are defined as:

$$f(x) = X(\omega) \tag{3.15}$$

$$g(x) = H(\omega)e^{j\omega t_x} \tag{3.16}$$

This allows Equation 3.12 to be expanded:

$$\frac{|y(t_x)|^2}{\overline{|n_o(t)|^2}} = \frac{\frac{1}{2\pi}\int |X(\omega)\mathrm{d}\omega|^2 \int |H(\omega)\mathrm{d}\omega|^2}{\frac{N_0}{4\pi}\int |H(\omega)|^2\mathrm{d}\omega} \tag{3.17}$$

Equation 3.17 can then be simplified:

$$\frac{|y(t_x)|^2}{\overline{|n_o(t)|^2}} = \frac{2}{N_0}\int |X(\omega)\mathrm{d}\omega|^2 \tag{3.18}$$

With these results, the special case of the Schwartz inequality given by Equation 3.14 is applied:

$$H(\omega)e^{j\omega t_x} = kX^*(\omega) \tag{3.19}$$

Rewriting:

$$H(\omega) = kX^*(\omega)e^{-j\omega t_x} \tag{3.20}$$

Applying the Fourier transform pairs:

$$X^*(\omega) \rightarrow x^*(-t) \tag{3.21}$$

$$X(\omega)e^{-j\omega t_x} \rightarrow x(t - t_x) \tag{3.22}$$

The final matched filter expression can be written as:

$$h(t) = kx^*(-t - t_x) \tag{3.23}$$

Equation 3.23 describes a filter whose response is identical to the received signal $v(t)$ but reversed in time and shifted by some amount t_x (k may be used to adjust the amplitude, but typically is left equal to 1). For the reverse time system, the received response is taken to be the basis pulse given by Equation 2.44. The resulting required matched filter response is shown in Figure 3.1 for $\beta = \ln(2)$.

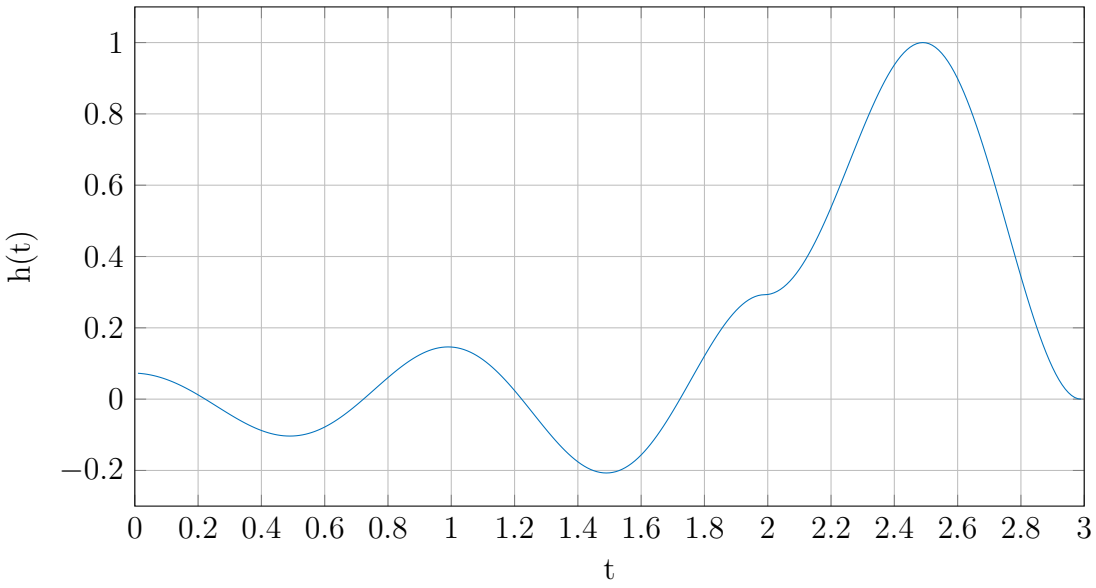

Figure 3.1: Reverse Time Matched Filter Response

3.1.2 Expected Performance

To demonstrate the expected performance of the matched filter, the following set of figures demonstrates its output graphically as the reverse time basis pulse is presented at its input. $\beta = \ln(2)$ was used for all numerical simulations. Figure 3.2 starts this process with the basis pulse partially shifted in to the matched filter.

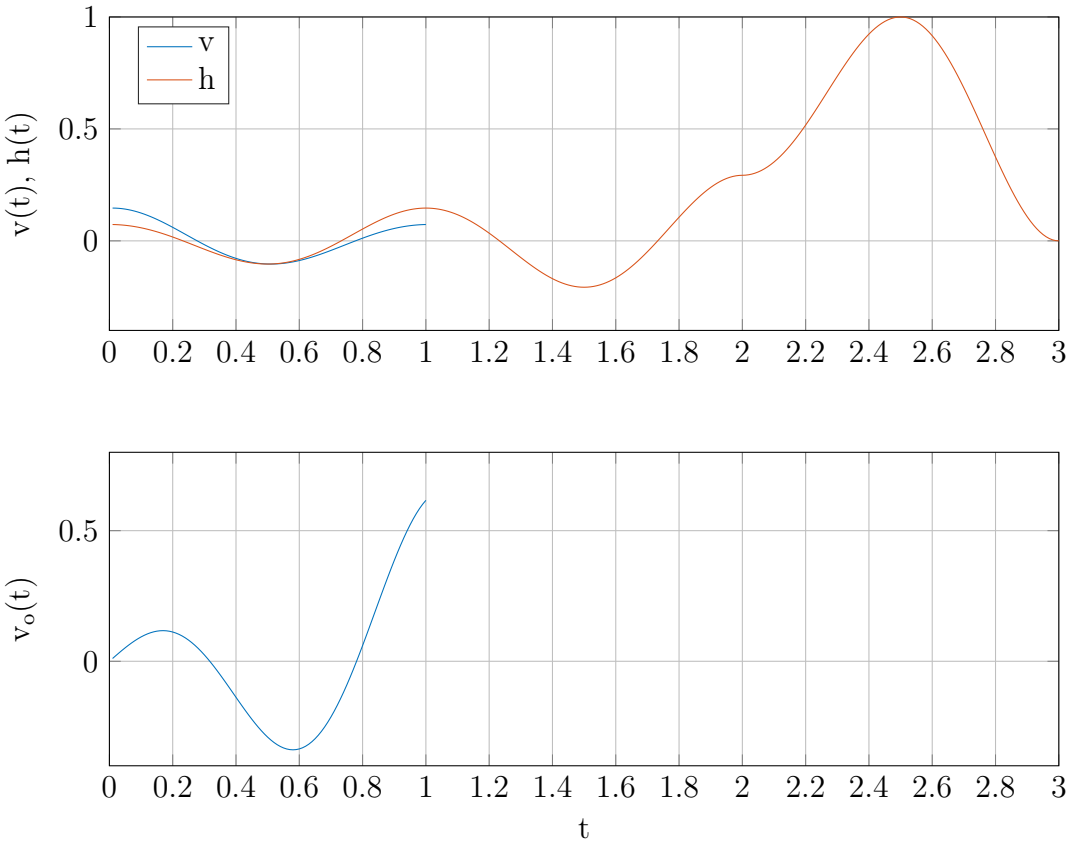

Figure 3.2: Matched Filter Performance (initial)

The top plot in this figure shows both the input (v, in blue) and the matched filter response (h, in orange). As expected, the response of the filter remains small since only a small segment of the basis pulse as been input. With more of the shift completed, as shown in Figure 3.3, the matched filter's response begins to increase.

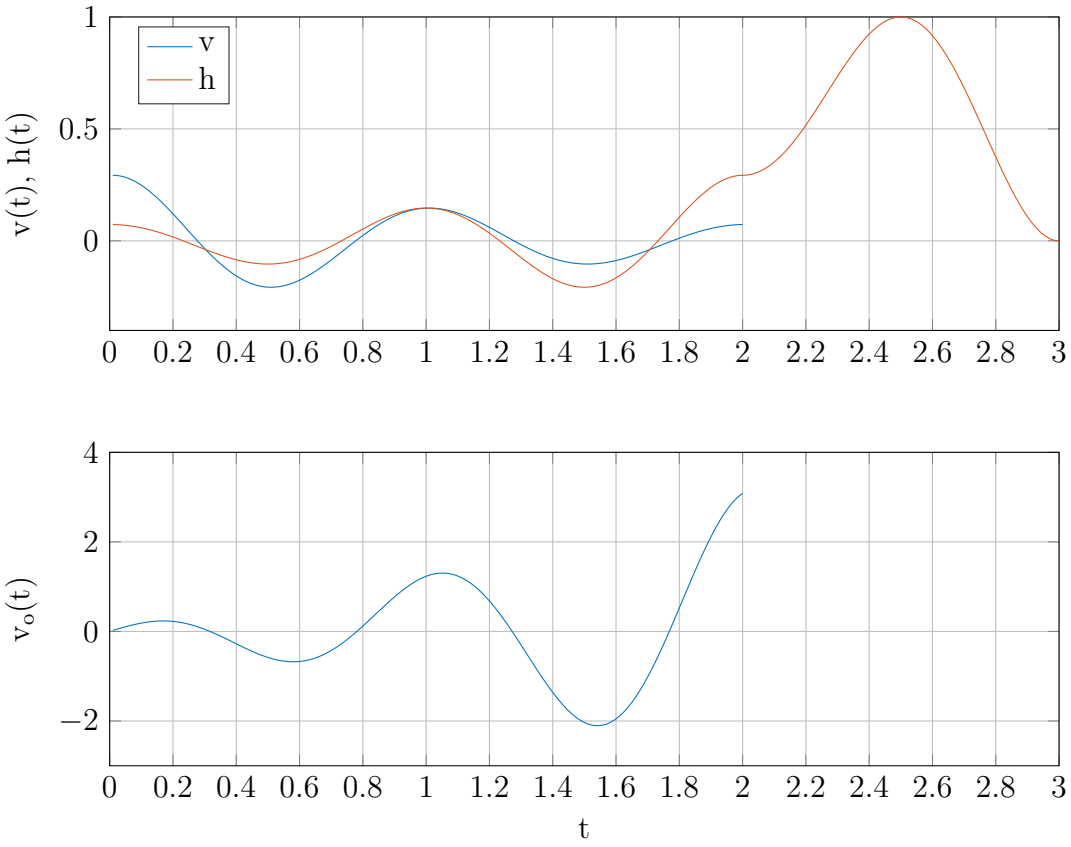

Figure 3.3: Matched Filter Performance (partial)

Even though the basis pulse is still only two thirds shifted in, the matched filter's output has increased by a factor of four. Finally, with the shift complete, the matched filter's output produces the expected sharp peak. This can be seen in Figure 3.4.

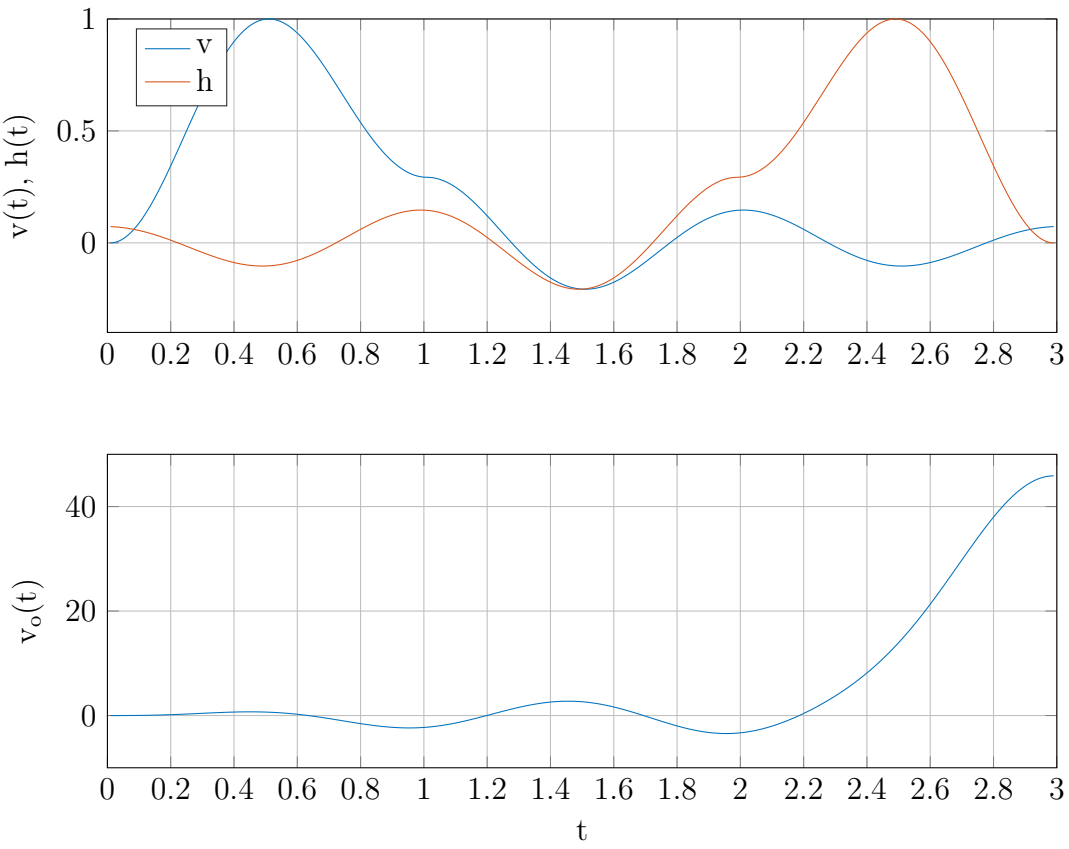

Figure 3.4: Matched Filter Performance (complete)

3.2 Numerical Model

3.2.1 Verification

A numerical model for the matched filter was constructed in MATLAB using the `filter` function. Using this model, the performance of the filter in a number of scenarios of interest was evaluated. The first test performed was to determine if the filter could properly detect the basis pulse when inserted in a random signal. Figure 3.5 demonstrates this process.

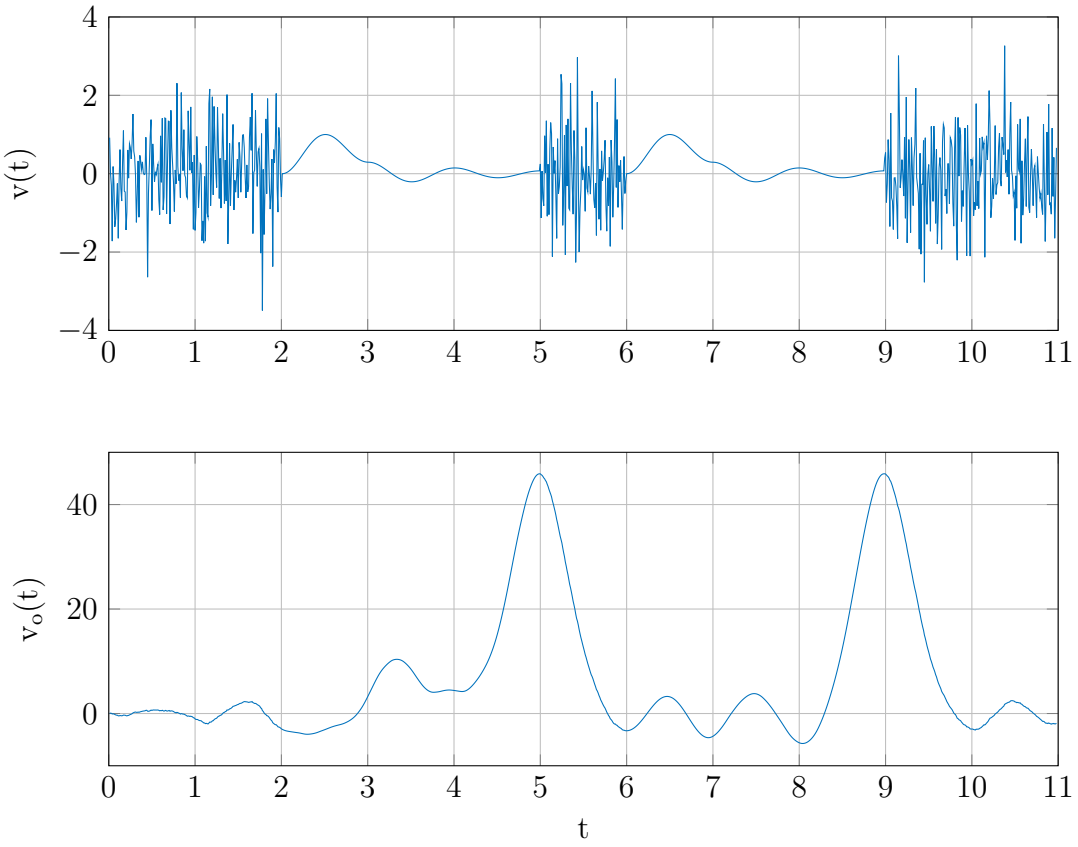

Figure 3.5: Matched Filter Response to Basis Pulse

The output from the numerical filter ($v_o(t)$) is shown in the bottom plot while the input ($v(t)$) is shown in the top. These results match closely with the performance seen in the previous tests and indicate that it correctly responds to the presence of a basis pulse. To verify the expected resilience in the presence of AWGN, a series of three additional tests was performed with increasing noise levels added for each successive run. MATLAB's `awgn` function was used to add noise to the existing model. Figures 3.6, 3.7, and 3.8 show these results with the SNR set to 10 dB, 3 dB, and 0 dB, respectively. In each figure, the segment containing the basis pulse with added noise is highlighted in red.

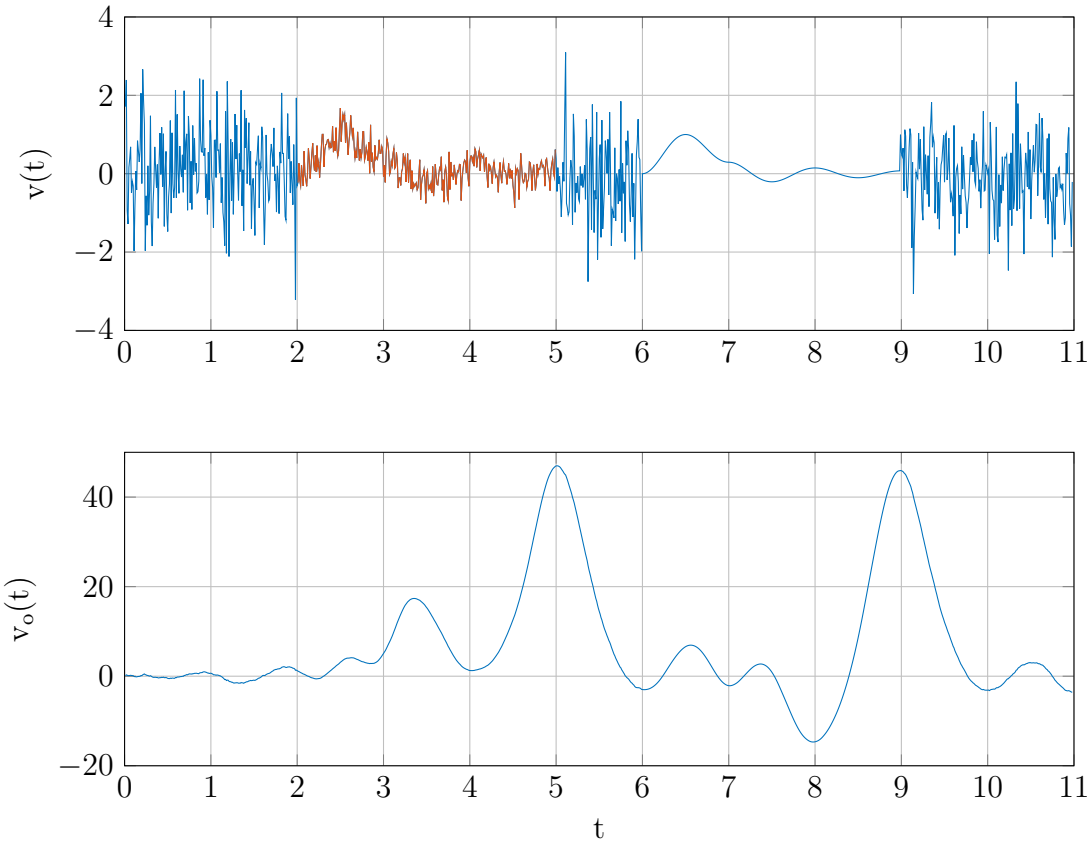

Figure 3.6: Matched Filter Response to Basis Pulse in Presence of AWGN (SNR=10 dB)

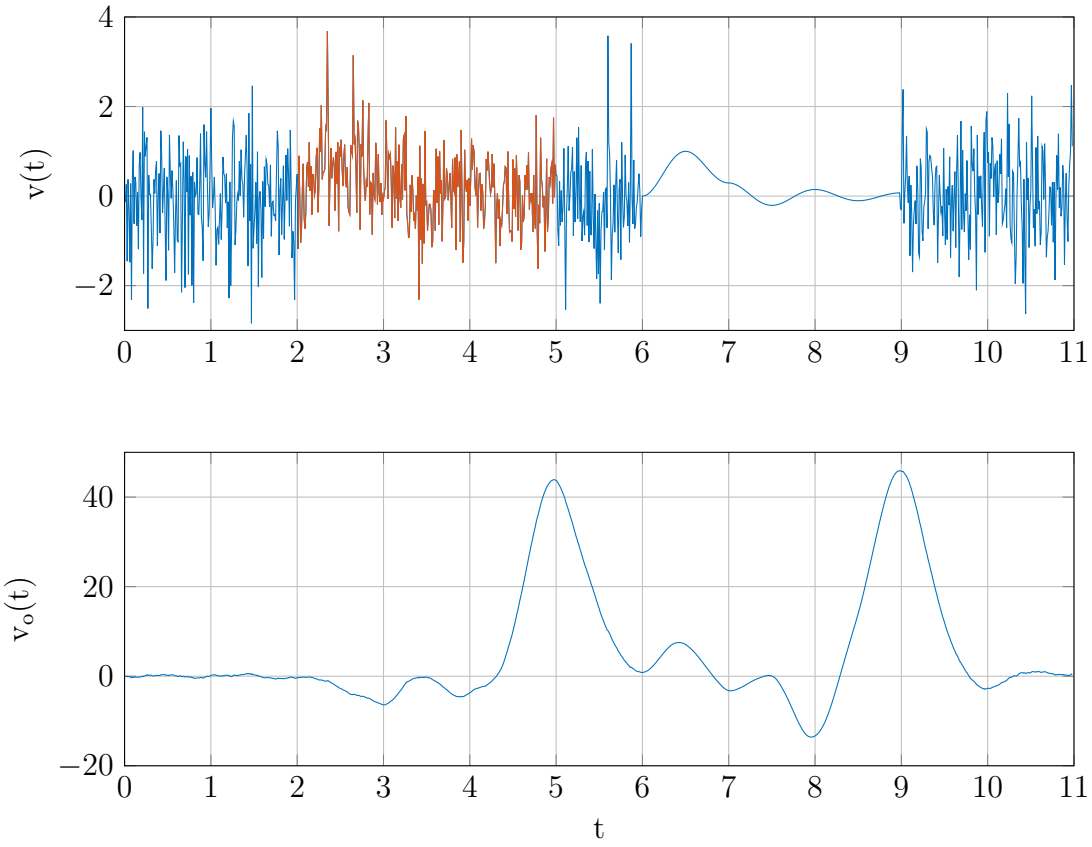

Figure 3.7: Matched Filter Response to Basis Pulse in Presence of AWGN (SNR=3 dB)

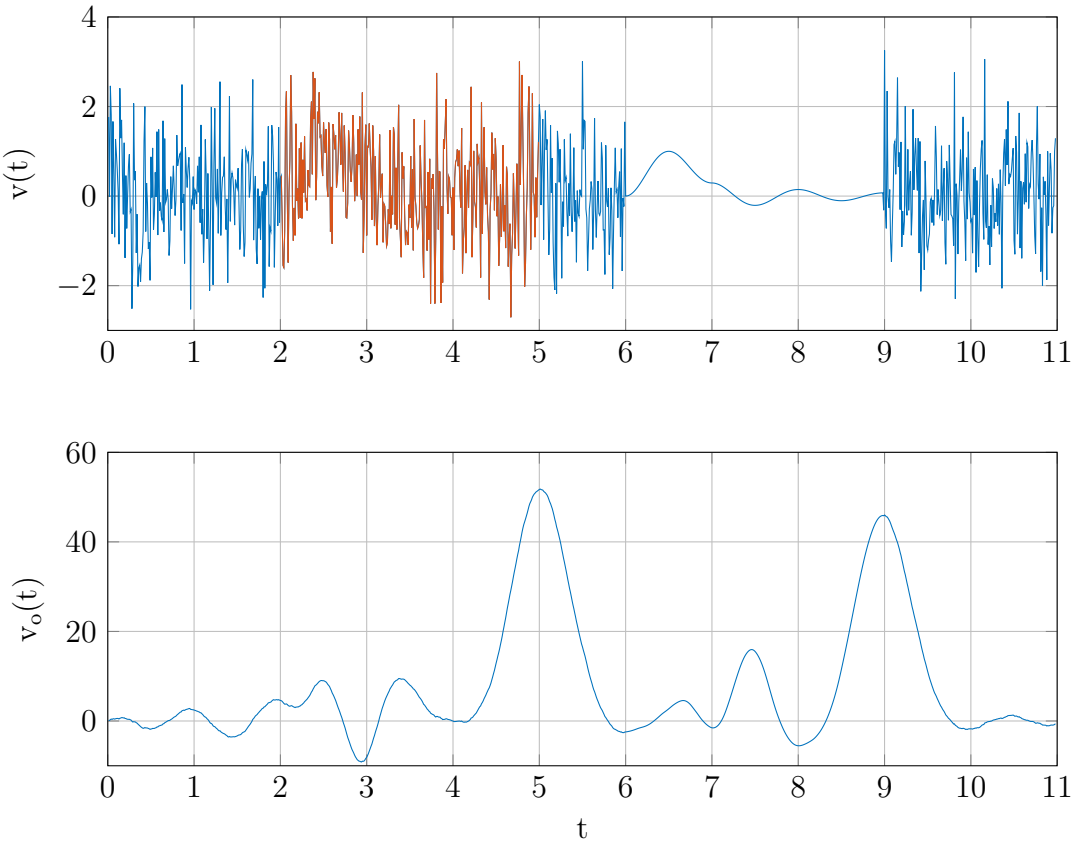

Figure 3.8: Matched Filter Response to Basis Pulse in Presence of AWGN (SNR=0 dB)

In all three cases where AWGN was present, the matched filter was able to correctly detect the presence of the basis pulse. Although the magnitude of the output peak varied slightly with the AWGN magnitude, it remained high enough to be readily detected for all inputs. The response to the random segments at the beginning and end of the input signal remained low throughout; this result combined with the peak behavior previously discussed indicates a strong selectivity for the basis pulse.

3.2.2 Chaotic Input

While the ability to detect a single basis pulse provides verification that the matched filter model is operating correctly, its true utility comes from the ability to detect a reverse time chaotic signal. The numerical model used above was tested with a chaotic signal inserted into a random waveform to verify this functionality. The output of this test is shown in the following figure.

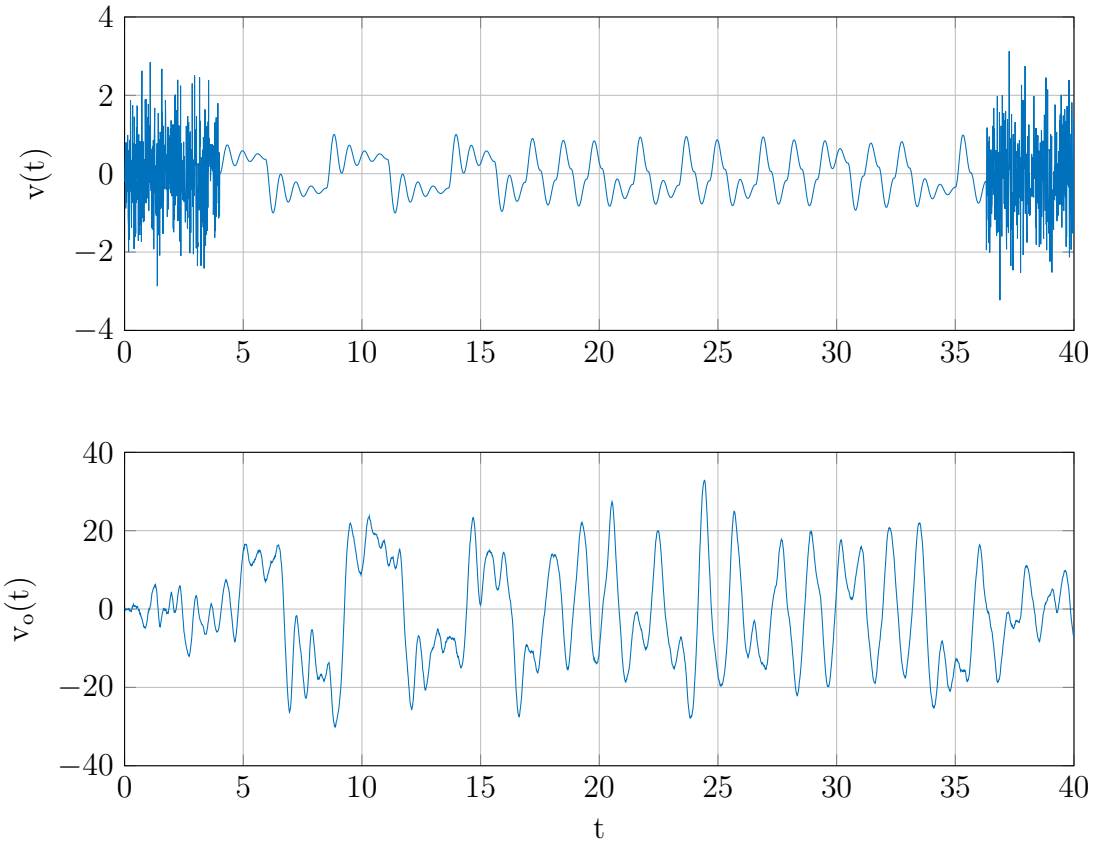

Figure 3.9: Matched Filter Response to Reverse Time Chaotic Waveform

Because the chaotic waveform consists of many instances of the basis pulse summed together, the matched filter shows a large number of peaks as the input is shifted through its response. The magnitude of these peaks is diminished as compared to the response when presented with a single instance of the basis pulse, but is still high enough to allow for reliable detection of the chaotic waveform.

The effect of AWGN present on the chaotic waveform has also been explored with the numerical matched filter. Rather than iterate through multiple noise levels, only the worst-case level (SNR = 0 dB) used above was examined. These results are presented in Figure 3.10.

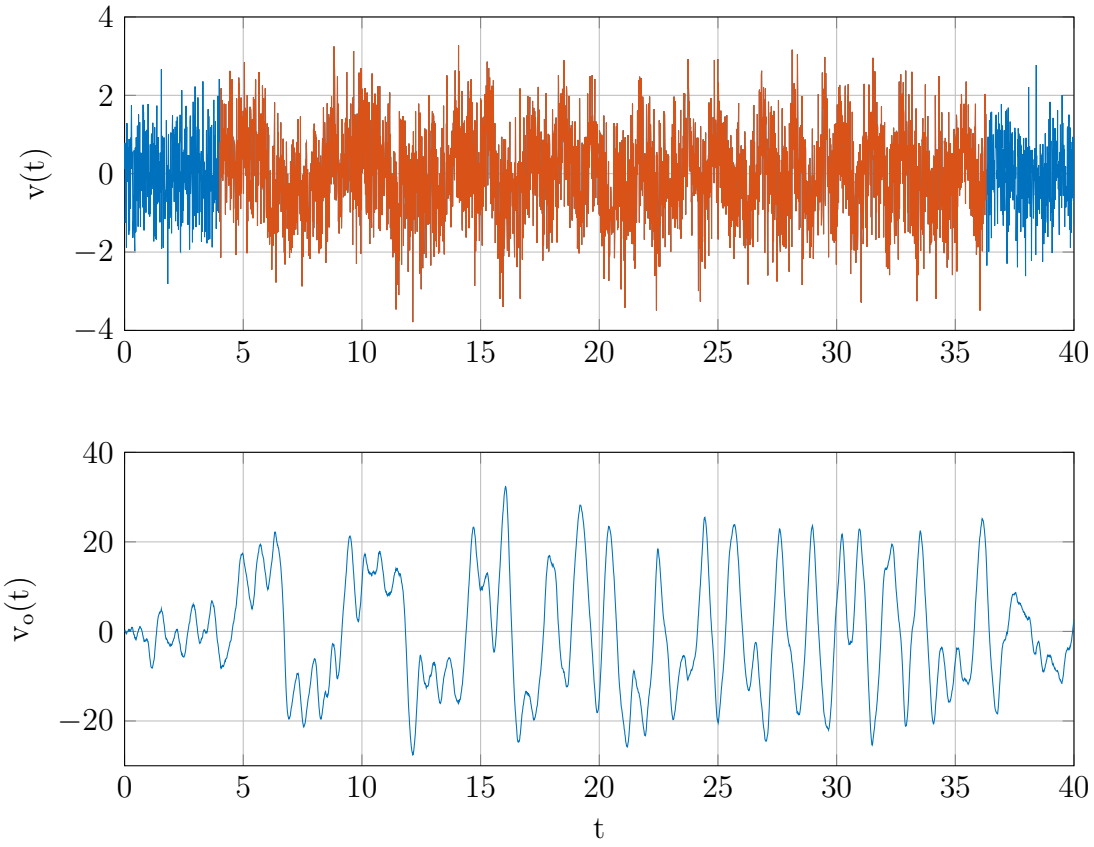

Figure 3.10: Matched Filter Response to Reverse Time Chaotic Waveform in Presence of AWGN (SNR=0 dB)

As with the basis pulse, even high magnitude AWGN does not significantly diminish the response of the matched filter to the presence of a chaotic waveform.

3.2.3 s_n Reconstruction

Limited post-processing can be applied to the matched filter response to reconstruct the s_n sequence used to generate the chaotic signal applied to its input. This post-processing requires the establishment of three threshold levels: a high threshold, a zero crossing or midpoint threshold, and a low threshold. Figure 3.11 demonstrates the placement of these levels where the high and low thresholds are denoted by the upper and lower red lines and the midpoint threshold is denoted by the black line.

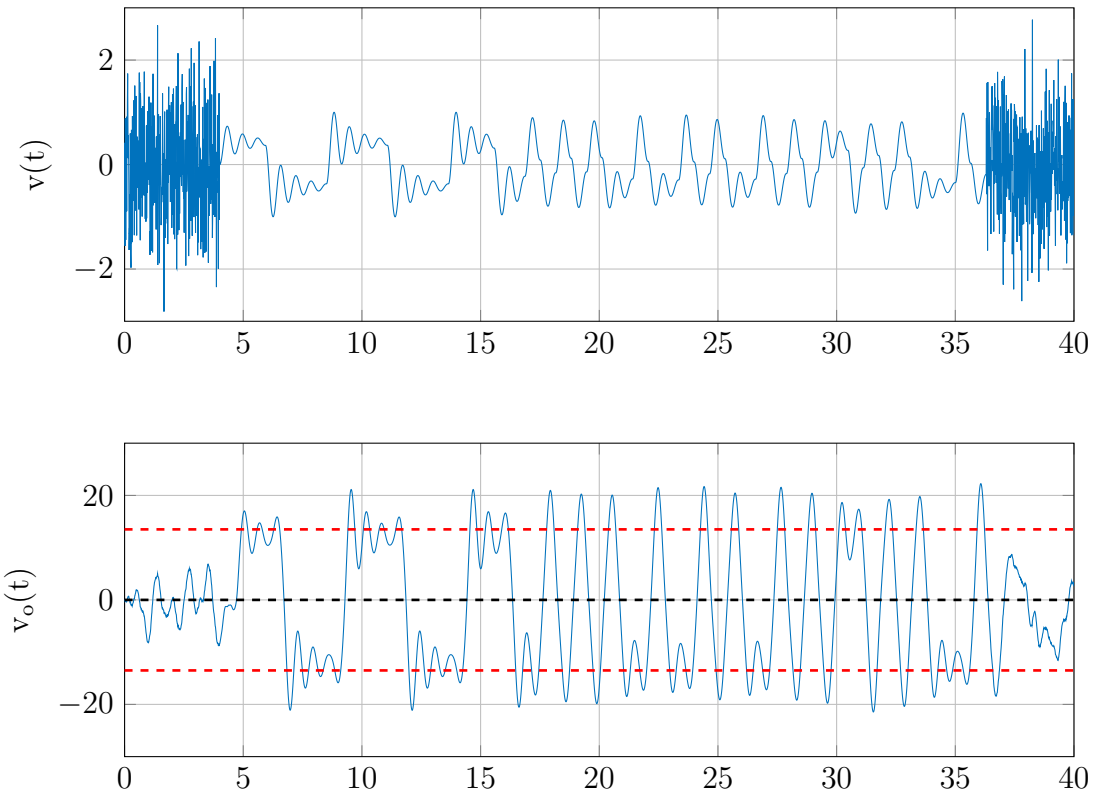

Figure 3.11: Threshold Levels Used for Reconstruction of s_n

New values of s_n always correspond to the matched filter output going above (or below) the high and low threshold values. Just detecting these events, however, does not properly reconstruct s_n - it only indicates the presence of the basis pulse (it is important to reiterate that the matched filter is matched to the basis pulse, not the overall chaotic waveform or s_n). The solution in Equation 2.45 allows for s_n to hold its value for many cycles, which results in many oscillations around the threshold. To account for this, the midpoint threshold is used to detect when a switch in the s_n value has occurred and then update the recovered s_n. This process is shown graphically in the following figure.

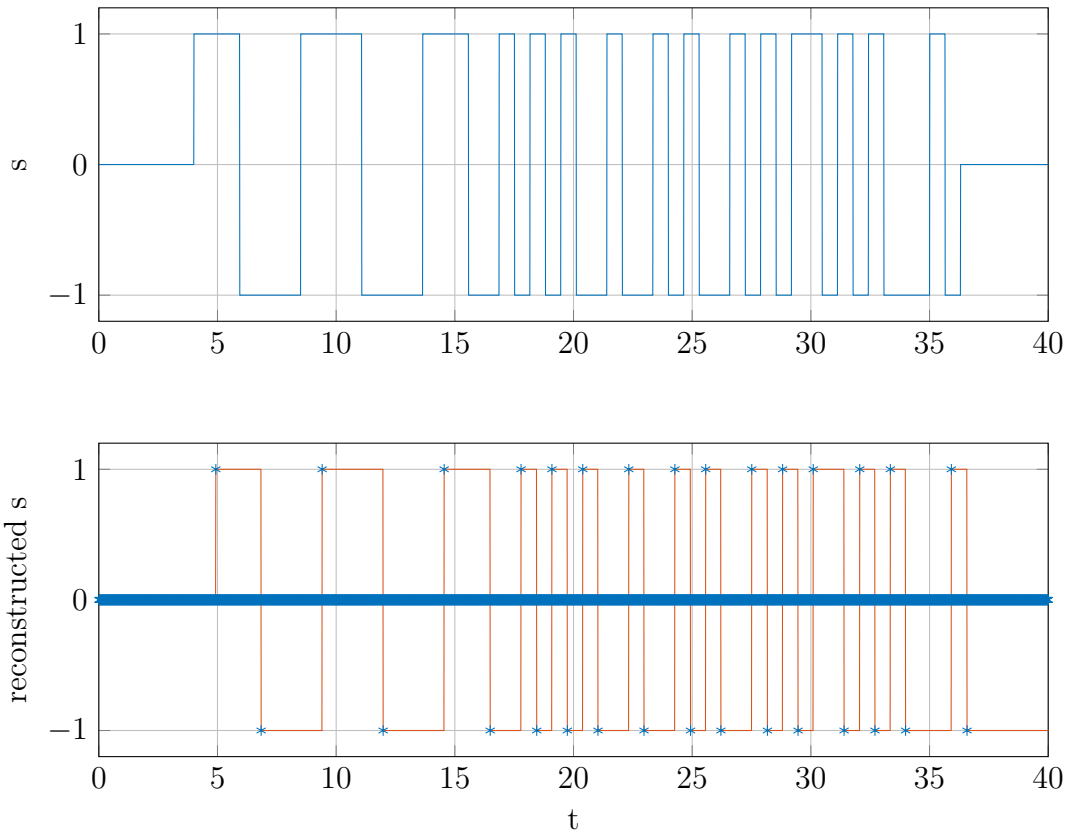

Figure 3.12: Reconstructed s_n

When compared to the original s state of the reverse time chaotic oscillator (top), the reconstructed s (bottom) appears identical apart from the time shift introduced by the matched filter operation. The blue asterisks marked on the reconstructed s plot indicate the decision made by the post processing algorithm at each time step. For the vast majority of the steps, this value is 0, corresponding to no change in s. When the matched filter output crosses through the midpoint threshold and then exceeds either the high or low threshold, however, the decision changes to indicate a new value. These decision points are then used to generate the overall reconstructed s as shown.

3.3 FIR Implementation

The numerical modeling in MATLAB provided a straightforward path for realization of the matched filter in hardware as a digital FIR filter. Values used for the filter's magnitude response can be used directly to define the coefficients necessary for the FIR algorithm. Any N^{th} order constant coefficient digital FIR function can be described as a convolution sum as shown in Equation 3.24 [47].

$$Y[n] = X[n] * H[n] = \sum_{k=0}^{N-1} H[k] * X[n-k] \tag{3.24}$$

The exact function performed is determined by the values of H, which are the FIR coefficients. This form can be implemented in hardware directly (and is often referred to as the direct-form FIR), but is generally less desirable due to its use of multipliers. One solution for addressing this shortcoming is using the sum of power of two (SOPOT) method to decompose multiplication into addition and multiplication by powers of two [48,49]. Equation 3.25 describes the process of determining the SOPOT representation of an arbitrary value $s(n)$.

$$s(n) = \sum_{i=0}^{J-1} a_{i,n} * 2^{b_{i,n}} \tag{3.25}$$

where $a_{i,n} \in \{-1, 0, 1\}$, $b_{i,n} \in \{0, 1, ..., u\}$, and J is the number of terms necessary to represent $s(n)$ [48]. The maximum value that can be represented by SOPOT is related to u, so its value determines the allowable range of coefficients. Because SOPOT contains only multiplication by powers of two, the multiplication functions can be replace by logical left shifts that shift by the $b_{i,n}$ values. An algorithm based on the work in [50] was developed in MATLAB to automate the calculation of these coefficients for any real number and is documented in the following listing.

```
function [a b] = sopot(h)
i = 1;

% continue iterating until all terms are generated
while h ~= 0
    % try to generate positive coeffecient first
    if h > 0
        a(i) = (abs(h)/h);
        b(i) = round(log2(abs(h)));
        h = h - 2^(b(i));
    % generate negative coefficient if needed
    else
        a(i) = (abs(h)/h);
        b(i) = round(log2(abs(h)));
        h = h + 2^(b(i));
    end
    i = i + 1;
end

% format results for display
out = '';
for i=1:length(a)
    out = sprintf('%s %+i*2^%i', out, a(i), b(i));
end
disp(out);
```

Listing 1: SOPOT Decomposition Algorithm in MATLAB

3.3.1 Architecture

A general block diagram of the FIR architecture is shown in Figure 3.13.

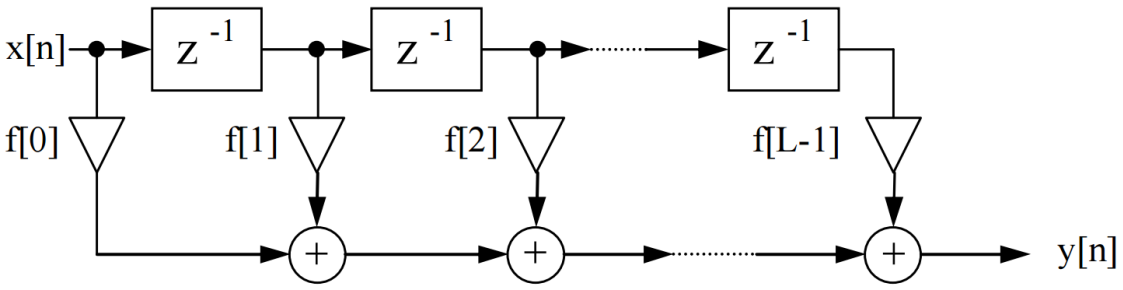

Figure 3.13: FIR Filter Block Diagram [2]

From this figure, it is apparent that the algorithm requires relatively few components for its hardware realization. The unit delay blocks (indicated as $z^{-}1$) map directly to D flip-flops where the clock period sets the delay time. The weights (denoted by $f[0]$-$f[L-1]$) require a varying number of adders and left shift circuits as indicated by the SOPOT decomposition. A Verilog example of a single unit from Figure 3.13 is provided in Listing 2.

```
1 // +1*2^5 +1*2^2 -1*2^0
2 adder_14b_3in a3(nw3, nd3 << 5, nd3 << 2, -(nd3 << 0));
3 adder_14b_2in af3(nf3, nw3, nf2);
4 wire [13:0] nf4;
5 wire [13:0] nd4;
6 wire [13:0] nw4;
7 d_ff_14b d4(nd3, clk, nd4);
```

Listing 2: Example SOPOT Form in Verilog HDL

Line 1 provides the polynomial describing the weight value for the block, while line 2 instantiates the adder used for the overall sum. Each of the adder's inputs consists of a single power of two implemented by the appropriate logical shift. The remaining lines contain a second adder to combine this block's results with the previous block as well as the D flip-flop. A MATLAB routine was written to generate and combine these blocks automatically given a set of FIR weights.

3.4 HDL Simulation

Making use of the Verilog code generated for the FIR filter, a HDL simulation was performed using ModelSim to determine expected performance of a physical implementation. The clock frequency used for this simulation significantly affected the output results; Figure 3.14 demonstrates this graphically.

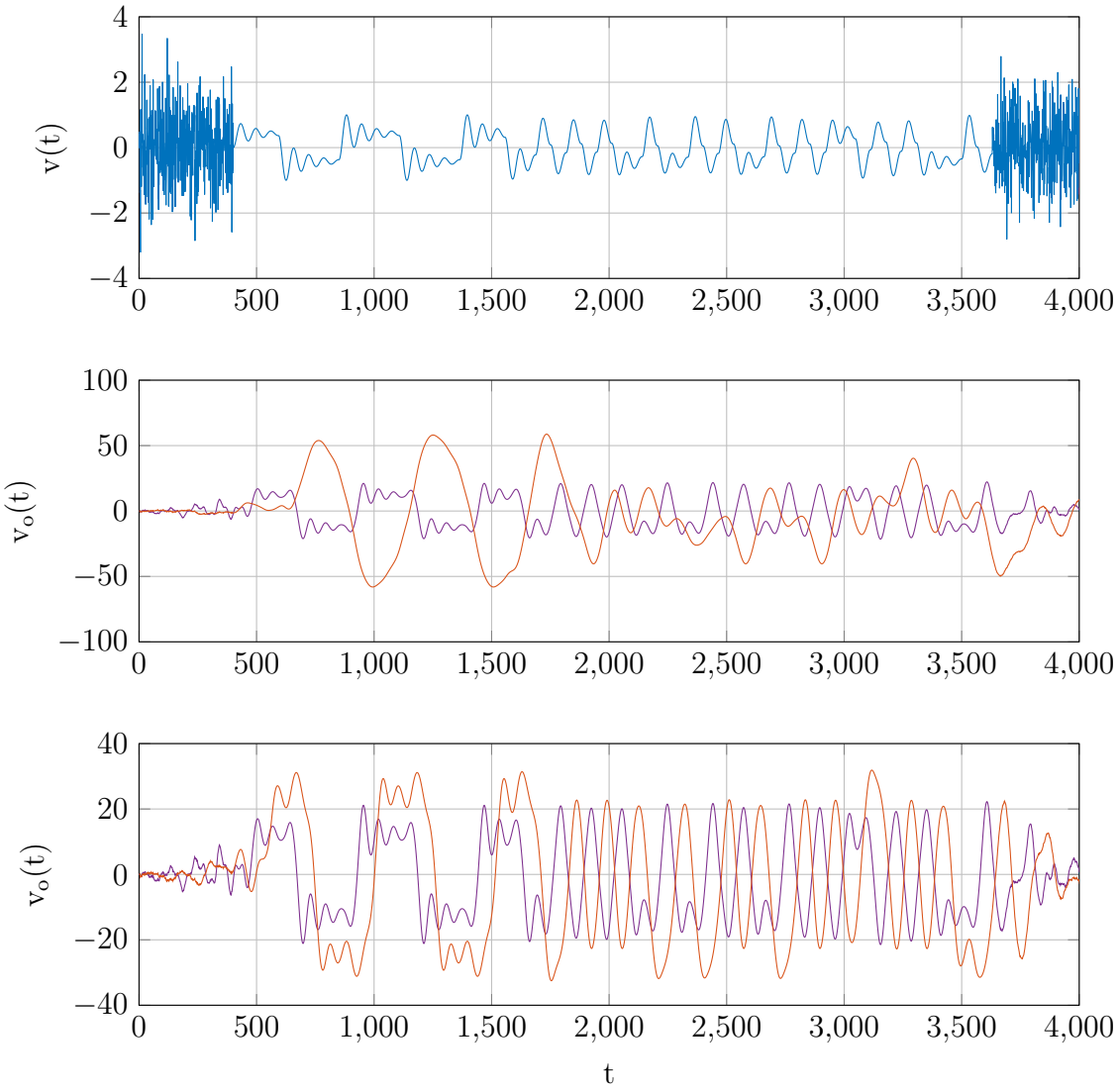

Figure 3.14: Effect of Frequency Shifts on Matched Filter Response

In the first plot, the chaotic input to the matched filter is shown; the desired output for this is shown in the following two plots in blue. Each of the red waveforms represents the matched filter output when the clock frequency is too fast (middle) and too slow (bottom). To achieve the desired output, the clock frequency must be set equal to the number of filter taps (coefficients) times the return time of the chaotic waveform so that the filter's shape matches the chaotic basis pulse.

Results of the HDL simulation indicate expected performance and are shown in the following figure.

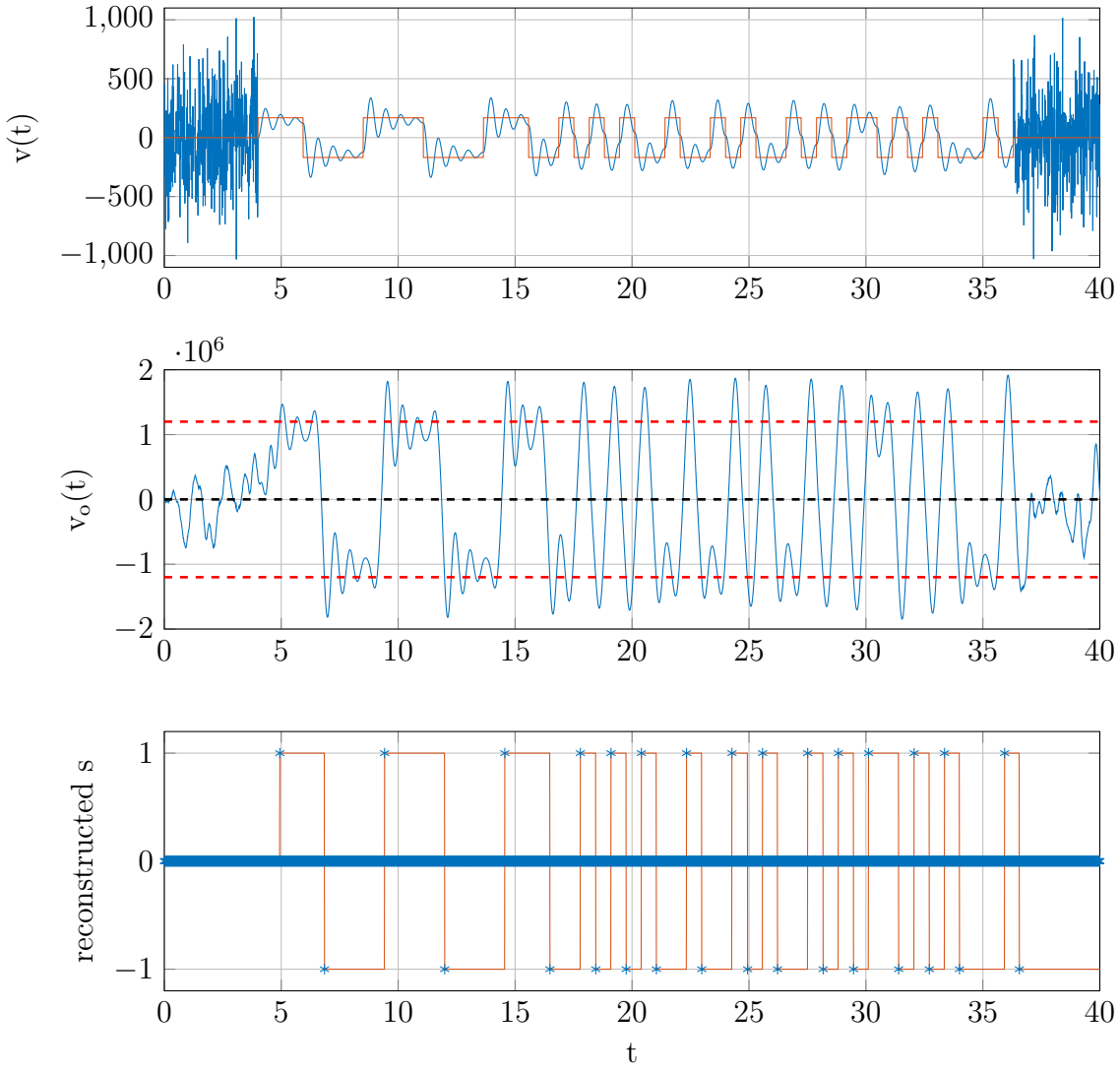

Figure 3.15: Simulation Results of FIR Matched Filter

The large scales used for the y axis on both the filter input and output result from converting the binary values generated from the simulation into decimal directly. 32 bit values were used for internal calculations to allow for high precision while 10 bit values were used for the input to correspond to typical ADC resolutions. The operating clock frequency was set to 180 MHz due to the use of the 1.8 MHz chaotic oscillator with 100 FIR coefficients.

Chapter 4

ASIC Components

To further improve upon the speed of chaotic systems, many (or all) of the components in such systems will need to be integrated into an ASIC. Initial work has been performed to evaluate the feasibility and potential performance of certain common analog chaotic system building blocks in IBM's state-of-the-art 9HP SiGe BiCMOS process. 9HP combines highly scaled 90 nm CMOS with extremely high performance NPN HBTs reaching a 300 GHz f_T [51].

4.1 Breakout Circuits

Rather than attempting to construct a complete chaotic system, multiple smaller test circuits were broken out individually so that each one could be thoroughly tested without having to compensate for the effects of other (undesired) components. Each breakout circuit was designed, simulated, and laid out using the Cadence software suite; Virtuoso XL was used for schematic entry and layout while Spectre was used for simulation. For testing purposes, each circuit contained its own power and ground rails and used 100 µm square pads for wire bonding.

4.1.1 FET Opamp

The first circuit developed was an opamp constructed using the 90 nm CMOS FETs. A FET design was chosen to allow for minimal area while still realizing acceptable gain and bandwidth. To minimize the overall complexity, the opamp bias current was designed to be provided off chip. Simulation results showing frequency response as a function of this bias current are shown in Figure 4.1.

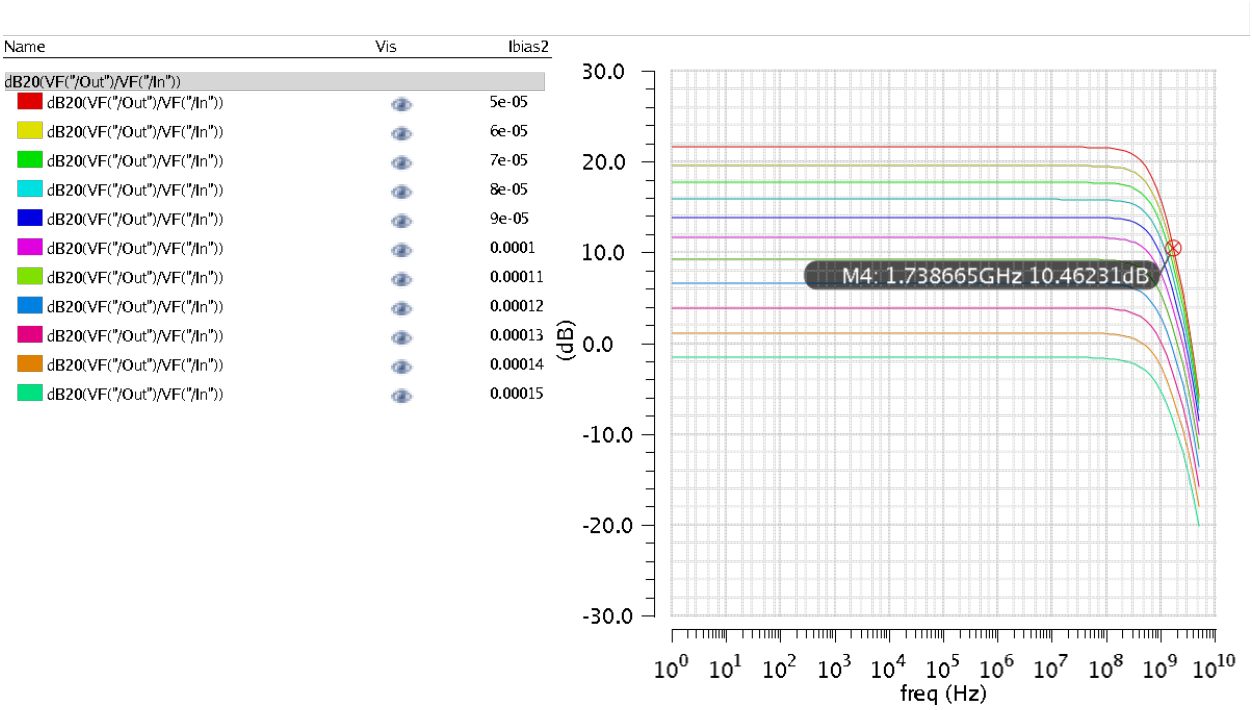

Figure 4.1: Opamp Simulation Results

The marker indicates the maximum gain occurs at a bias current of 50 µm. 1.73 GHz was chosen as the frequency of interest to correspond with the Colpitts oscillator discussed in the next section. A schematic of the circuit as fabricated in included on the following page. A typical three stage design was used: the first stage provides differential input with an active load, the second provides gain, and the third acts as a buffer to provide a voltage output.

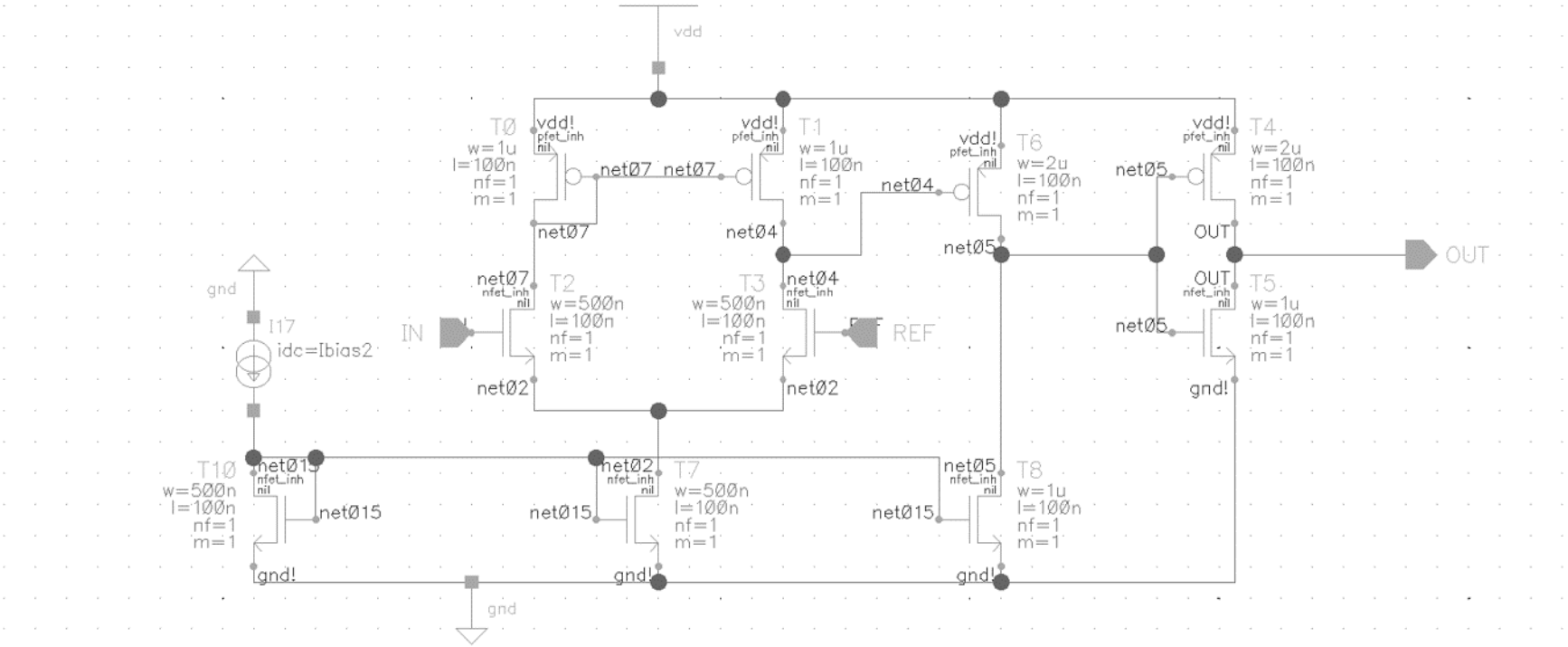

Figure 4.2: Opamp Schematic

4.1.2 Colpitts Oscillator

A Colpitts oscillator was also designed and included as a breakout circuit. While not directly suitable for inclusion in an exactly solvable chaotic system directly, this oscillator allows for characterization of the frequency performance of both the passive devices that form its tank as well as the transistor that provides its gain. Direct chaos generation - albeit without a closed form solution - is also possible by carefully tuning the bias current [52, 53]. To facilitate testing, both the bias current and the reference level of the tank were brought out to pads so they could be controlled off chip. Component values were determined using equations 4.1 and 4.2 for a target frequency f_0 of 1.7 GHz [54]:

$$f_0 = \frac{1}{2\pi}\sqrt{\frac{1}{C_p L}} \tag{4.1}$$

$$C_p = \frac{C_1 C_2}{C_1 + C_2} \tag{4.2}$$

For C_p set to 1 pF, an L of 8.8 nH was needed. A schematic showing the final circuit design as well as simulation results confirming (non chaotic) operation are presented in the following figures.

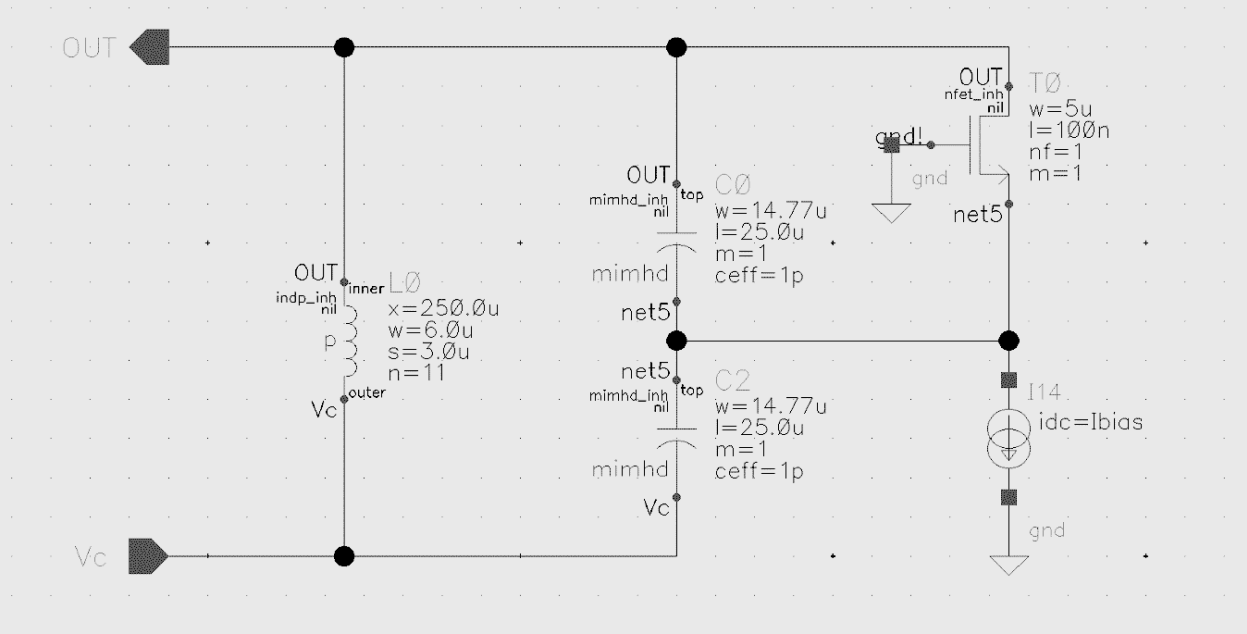

Figure 4.3: Colpitts Oscillator Schematic

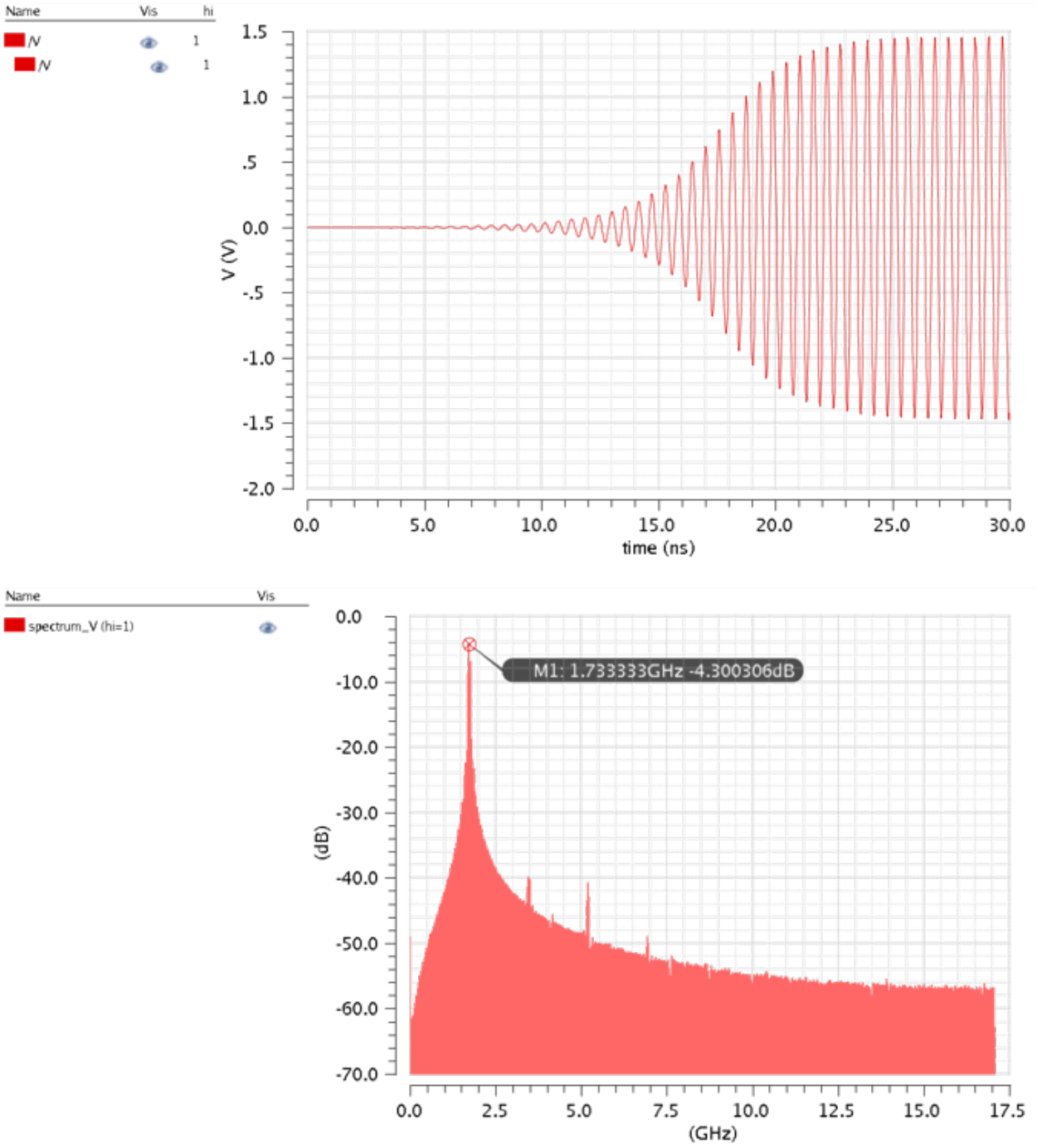

Figure 4.4: Colpitts Oscillator Simulation Results (top: time, bottom: spectrum)

4.1.3 Summary

Multiple breakout circuits were fabricated in the IBM 9HP process - the two discussed occupied a total die area of approximately 0.5 mm^2 and required 11 total pads. A summary of the area and number of pads for each circuit is provided in the following table.

Subcircuit	**Dimensions**	**Area**	**Pads**
FET opamp	568 µm x 341 µm	0.194 mm^2	6
Colpitts oscillator	446 µm x 651 µm	0.291 mm^2	5

Table 4.1: Performance Summary

Due to significant delays from the foundry, finished die were not received in time to include hardware test results. A photo of the die is included on the following page; the chaos breakout circuits are contained in the red (opamp) and green (Colpitts oscillator) outlines on the lower left corner of the die. The following table lists the pinout for each device.

FET opamp		**Colpitts oscillator**	
Pin Number	**Function**	**Pin Number**	**Function**
1	VDD	1	VDD
2	Out	2	Vout
3	GND	3	GND
4	In+	4	Ibias
5	In-	5	Vc
6	Ibias		

Table 4.2: 9HP Die Pinout

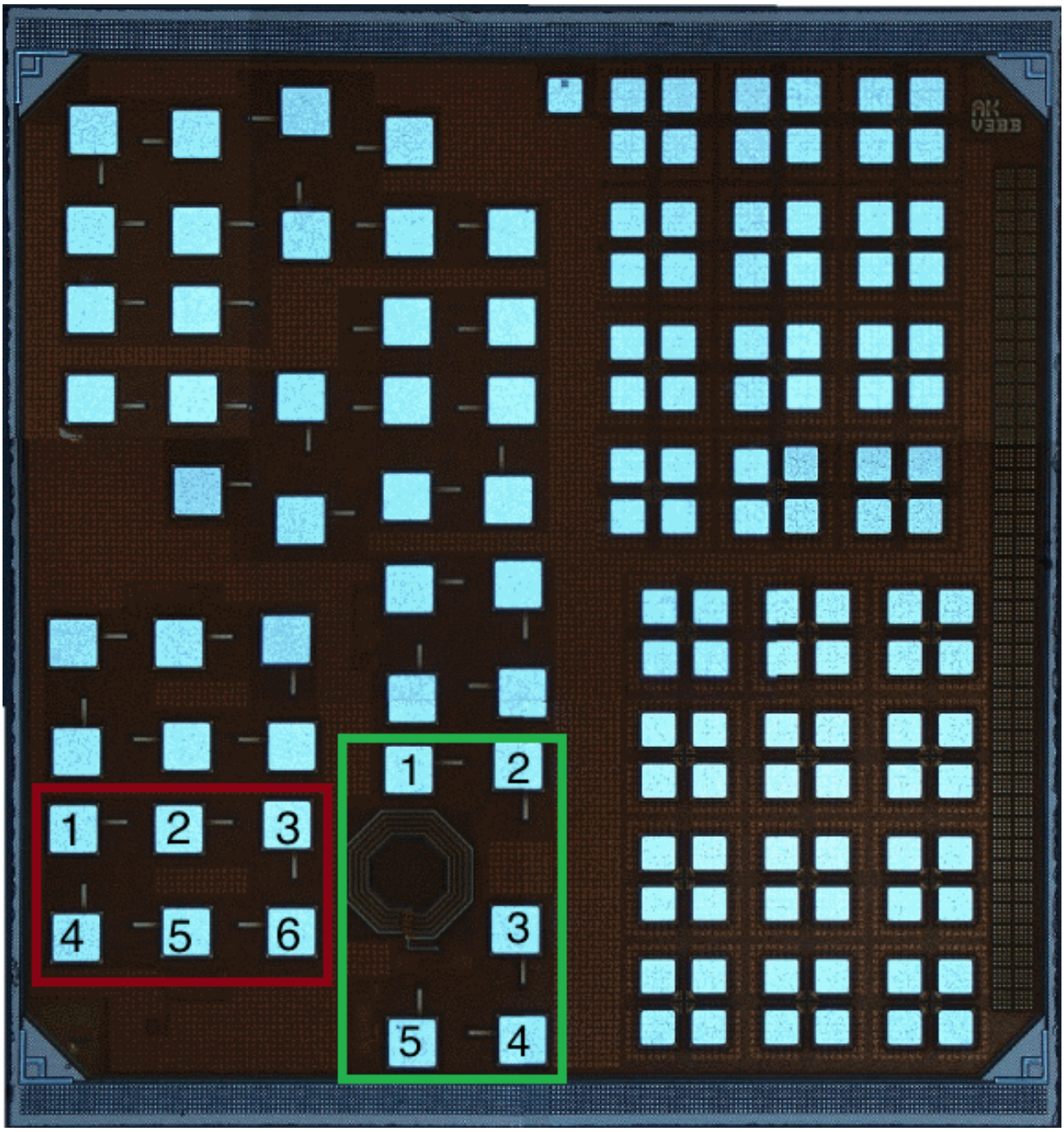

Figure 4.5: ASIC Die Photo

Chapter 5

Conclusions and Future Work

Reverse time chaos provides a solution for generating chaotic waveforms at speeds higher than have previously been realized with relatively simple hardware. It has shown that driving a passive RLC circuit with appropriate signal sequences from a chaotic iterated map will produce reverse time chaos that is in close agreement with numerical calculations. Possibilities for precise control and grammar restriction arising from the use of two different chaotic maps have been discussed.

Both simulation and hardware results for the reverse time chaotic system have been introduced and demonstrated to operate correctly at 1.8 MHz. The analog circuit used for this system was shown to require a low number of components, all of which are suitable for integration. HDL code for the digital portion of the circuit was synthesized and run in hardware on a FPGA; this verified that the HDL is ready for synthesis and inclusion on an ASIC.

In addition, matched filter decoding has been shown to provide a viable means of detection of reverse time chaotic waveforms in the presence of even high magnitude AWGN. The ability to both detect the waveform and use this detection to reconstruct the original s sequence has been demonstrated. Conversion to a FIR filter has been performed and implemented in HDL; simulation results of this code indicate that a hardware realization is possible and likely to function correctly using the generated HDL code.

Preliminary test circuits were designed and successfully taped out into a state-of-the-art SiGe process. Simulation of these components provided preliminary performance estimates of operating speeds exceeding 1 GHz. Final die have been received and are in preparation for hardware testing to confirm these results.

Overall, this implementation has been demonstrated as a viable means to generate high frequency chaotic waveforms that could be utilized in a wide range of applications. For chaotic waveform generation, neither digital to analog conversion nor complex analog circuitry is required; while for matched filter decoding, no analog circuitry is required at all. Both components are readily scalable to much higher operating frequencies without requiring any significant circuit modifications.

To realize these higher operating frequencies, multiple paths are open for future work in this area. Frequencies between 10 MHz and 50 MHz are readily achievable using the PCB and FPGA combination presented in this work; synthesis estimates of the digital iterated maps indicate correct operation is possible on the relatively low speed Cyclone IV-E board up to 25 MHz. Scaling to even higher operating frequencies will require ASIC implementations, but the preliminary ASIC results shown above provide a strong starting point for such development.

Bibliography

[1] M. P. Kennedy, "Three steps to chaos. I. Evolution," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 40, no. 10, pp. 640–656, 1993. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=246140

[2] C. Damian and E. Lunca, "A low area FIR filter for FPGA implementation." IEEE, Aug. 2011, pp. 521–524. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6043675

[3] K. Umeno, "Method of constructing exactly solvable chaos," *Physical Review E*, vol. 55, no. 5, p. 5280, 1997.

[4] N. J. Corron, J. N. Blakely, and M. T. Stahl, "A matched filter for chaos," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 20, no. 2, pp. 023 123–023 123–10, Jun. 2010. [Online]. Available: http://chaos.aip.org/resource/1/chaoeh/v20/i2/p023123_s1

[5] N. J. Corron and J. N. Blakely, "Exact folded-band chaotic oscillator," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 2, pp. 023 113–023 113–7, Apr. 2012. [Online]. Available: http://chaos.aip.org/resource/1/chaoeh/v22/i2/p023113_s1?isAuthorized=no

[6] C. Grebogi and E. Ott, "Communicating with chaos," 1993. [Online]. Available: http://clm.utexas.edu/compjclub/papers/Hayes1993.pdf

[7] S. T. Hayes, "Chaos from linear systems: implications for communicating with chaos, and the nature of determinism and randomness," *Journal of Physics: Conference Series*, vol. 23, pp. 215–237, Jan. 2005. [Online]. Available: http://stacks.iop.org/1742-6596/23/i=1/a=024?key=crossref.e6d25dae61e91a70b26f83c4308b4308

[8] G. Heidari-Bateni and C. McGillem, "A chaotic direct-sequence spread-spectrum communication system," *Communications, IEEE Transactions on*, vol. 42, no. 234, pp. 1524–1527, 1994.

[9] Y. Hirata and K. Judd, "Constructing dynamical systems with specified symbolic dynamics," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 15, no. 3, p. 033102, 2005. [Online]. Available: http://scitation.aip.org/content/aip/journal/chaos/15/3/10.1063/1.1944467

[10] S. Hayes, C. Grebogi, E. Ott, and A. Mark, "Experimental control of chaos for communication," *Physical Review Letters*, vol. 73, no. 13, pp. 1781–1784, Sep. 1994. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.73.1781

[11] S. D. Cohen and D. J. Gauthier, "A pseudo-matched filter for chaos," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 3, p. 033148, 2012. [Online]. Available: http://scitation.aip.org/content/aip/journal/chaos/22/3/10.1063/1.4754437

[12] C.-C. Chen and K. Yao, "Stochastic-calculus-based numerical evaluation and performance analysis of chaotic communication systems," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 12, pp. 1663–1672, Dec. 2000.

[13] J. N. Blakely, D. W. Hahs, and N. J. Corron, "Communication waveform properties of an exact folded-band chaotic oscillator," *Physica D: Nonlinear Phenomena*, vol. 263, pp. 99–106, Nov. 2013. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0167278913002509

[14] G. Kolumban, "Theoretical noise performance of correlator-based chaotic communications schemes," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 12, pp. 1692–1701, Dec. 2000.

[15] Z. Liu, X. Zhu, W. Hu, and F. Jiang, "Principles of chaotic signal radar," *International Journal of Bifurcation and Chaos*, vol. 17, no. 05, pp. 1735–1739, May 2007. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218127407018038

[16] T. L. Carroll, "Optimizing chaos-based signals for complex radar targets," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 17, no. 3, p. 033103, 2007. [Online]. Available: http://scitation.aip.org/content/aip/journal/chaos/17/3/10.1063/1.2751392

[17] G. M. Hall, E. J. Holder, S. D. Cohen, and D. J. Gauthier, "Low-cost chaotic radar design," K. I. Ranney and A. W. Doerry, Eds., May 2012, pp. 836 112–836 112–13. [Online]. Available: http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1355084

[18] A. Tsonis and J. Elsner, "Chaos, strange attractors, and weather." *Bulletin of the American Meteorological Society*, vol. 70, pp. 14–23, 1989.

[19] H. Leung, "Applying chaos to radar detection in an ocean environment: an experimental study," *Oceanic Engineering, IEEE Journal of*, vol. 20, no. 1, pp. 56–64, 1995.

[20] S. Nakagawa and T. Saito, "An RC OTA hysteresis chaos generator," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 43, no. 12, pp. 1019–1021, Dec. 1996.

[21] T. Saito and H. Fujita, "Chaos in a manifold piecewise linear system," *Electronics and Communications in Japan (Part I: Communications)*, vol. 64, no. 10, pp. 9–17, 1981.

[22] B. Muthuswamy, "Implementing memristor based chaotic circuits," *International Journal of Bifurcation and Chaos*, vol. 20, no. 05, pp. 1335–1350, May 2010. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218127410026514

[23] A. Buscarino, L. Fortuna, M. Frasca, and L. Valentina Gambuzza, "A chaotic circuit based on Hewlett-Packard memristor," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 22, no. 2, p. 023136, 2012. [Online]. Available: http://link.aip.org/link/CHAOEH/v22/i2/p023136/s1&Agg=doi

[24] A. Beal, J. Bailey, S. Hale, R. Dean, M. Hamilton, J. Tugnait, D. Hahs, and N. Corron, "Design and simulation of a high frequency exact solvable chaotic oscillator," in *Military Communications Conference, 2012 - Milcom 2012*, 2012, pp. 1–6.

[25] N. J. Corron, S. T. Hayes, S. D. Pethel, and J. N. Blakely, "Chaos without nonlinear dynamics," *Phys. Rev. Lett.*, vol. 97, no. 2, p. 024101, Jul. 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevLett.97.024101

[26] S. H. Strogatz, *Nonlinear dynamics and chaos: with application to physics, biology, chemistry, and engineering*. Cambridge: Westview press, 2000.

[27] R. L. Devaney, *A first course in chaotic dynamical systems: theory and experiment*, ser. Studies in nonlinearity. Reading, Mass: Addison-Wesley, 1992.

[28] E. N. Lorenz, "Deterministic nonperiodic flow," *Journal of the Atmospheric Sciences*, vol. 20, no. 2, pp. 130–141, Mar. 1963. [Online]. Available: http://journals.ametsoc.org/doi/abs/10.1175/1520-0469%281963%29020%3C0130%3ADNF%3E2.0.CO%3B2

[29] O. Rössler, "An equation for continuous chaos," *Physics Letters A*, vol. 57, no. 5, pp. 397–398, Jul. 1976. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0375960176901018

[30] R. Brown, "Generalizations of the Chua equations," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 40, no. 11, pp. 878–884, Nov. 1993.

[31] M. Itoh and L. O. Chua, "Memristor oscillators," *International Journal of Bifurcation and Chaos*, vol. 18, no. 11, pp. 3183–3206, 2008. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0218127408022354

[32] K. O'Donoghue, M. P. Kennedy, and P. Forbes, "A fast and simple implementation of Chua's oscillator using a cubic–like chua diode," in *Circuit Theory and Design, 2005. Proceedings of the 2005 European Conference on*, vol. 2, 2006, pp. II–83. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1522998

[33] N. J. Corron and J. N. Blakely, "A Matched Filter For Communicating With Chaos," *AIP Conference Proceedings*, vol. 1339, no. 1, pp. 25–35, Apr. 2011. [Online]. Available: http://proceedings.aip.org/resource/2/apcpcs/1339/1/25_1?isAuthorized=no

[34] J. P. Bailey, "Simulation and characterization of an exact solvable chaotic oscillator and matched filter," Master's thesis, Auburn University, 2012.

[35] W. Weckesser, "One Dimensional Maps," Colgate University, Feb. 2005. [Online]. Available: http://math.colgate.edu/~wweckesser/math312Spring05/handouts/Maps1D.pdf

[36] H. Zhao, H. Kwan, and J. Yu, "Fractional discrete-time chaotic map," in *2006 IEEE International Symposium on Circuits and Systems, 2006. ISCAS 2006. Proceedings*, May 2006, pp. 4 pp.–.

[37] A. Beirami, H. Nejati, and W. Ali, "Zigzag map: a variability-aware discrete-time chaotic-map truly random number generator," *Electronics Letters*, vol. 48, no. 24, pp. 1537–1538, Nov. 2012.

[38] J. P. Crutchfield and N. H. Packard, "Symbolic dynamics of one-dimensional maps: Entropies, finite precision, and noise," *International Journal of Theoretical Physics*, vol. 21, no. 6-7, pp. 433–466, 1982. [Online]. Available: http://link.springer.com/article/10.1007/BF02650178

[39] S. G. Williams, "Introduction to symbolic dynamics," in *Proceedings of Symposia in Applied Mathematics*, vol. 60, 2004, pp. 1–12. [Online]. Available: http://www.southalabama.edu/mathstat/personal_pages/williams/wilshort.pdf

[40] N. Corron, S. Hayes, S. Pethel, and J. Blakely, "Synthesizing folded band chaos," *Physical Review E*, vol. 75, no. 4, Apr. 2007. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.75.045201

[41] J. Bailey, A. Beal, J. Tugnait, R. Dean, and M. Hamilton, "High-frequency reverse-time chaos generation using digital chaotic maps," *Electronics Letters*, vol. 50, no. 23, pp. 1683–1685, Nov. 2014. [Online]. Available: http://digital-library.theiet.org/content/journals/10.1049/el.2014.2709

[42] *LTSpiceIV Manual*, LTspiceHelp.chm, Linear Technology, Mar. 2015. [Online]. Available: http://cds.linear.com/docs/ltspice

[43] *LT1220 45MHz, 250V/us Operational Amplifier*, 1220fb.pdf, Linear Technology, 1991. [Online]. Available: http://http://cds.linear.com/docs/Datasheet/

[44] A. J. Wilkinson, "2.7 Matched Filter," University of Cape Town, Apr. 2014. [Online]. Available: http://local.eleceng.uct.ac.za/courses/EEE3086F/notes/212-Matched_Filter_2up.pdf

[45] E. Kamen and B. Heck, *Fundamentals of Signals and Systems: With MATLAB Examples.* Prentice Hall PTR, 2000.

[46] S. S. Haykin, *Communication systems*, 4th ed. New York: Wiley, 2001.

[47] F. Nekoei, Y. Kavian, and O. Strobel, "Some schemes of realization digital fir filters on fpga for communication applications," in *Microwave and Telecommunication Technology (CriMiCo), 2010 20th International Crimean Conference*, sept. 2010, pp. 616 –619.

[48] K. Yeung and S. Chan, "Multiplier-less fir digital filters using programmable sum-of-power-of-two (sopot) coefficients," in *Field-Programmable Technology, 2002. (FPT). Proceedings. 2002 IEEE International Conference on*, dec. 2002, pp. 78 – 84.

[49] C. Li, "Design and realization of fir digital filters based on matlab," in *Anti-Counterfeiting Security and Identification in Communication (ASID), 2010 International Conference on*, july 2010, pp. 101 –104.

[50] J. K. Dhobi, D. Y. B. Shukla, and D. K. R. Bhatt, "FPGA Implementation of FIR Filter using Various Algorithms: A Retrospective," *International Journal of Research in Computer Science*, vol. 4, no. 2, pp. 19–24, Mar. 2014. [Online]. Available: http://www.ijorcs.org/manuscript/id/81/doi:10.7815/ijorcs.42.2014.081/jinalkumari-k-dhobi/fpga-implementation-of-fir-filter-using-various-algorithms-a-retrospective

[51] "IBM Preps New Wireless Chip Technology," Jun. 2013. [Online]. Available: http://www-03.ibm.com/press/us/en/pressrelease/41224.wss

[52] M. Kennedy, "Chaos in the colpitts oscillator," *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, vol. 41, no. 11, pp. 771–774, Nov 1994.

[53] L. Jingxia and M. Fuchang, "Chaos generation in microwave band using improved colpitts oscillator," in *Computing, Measurement, Control and Sensor Network (CMCSN), 2012 International Conference on*, July 2012, pp. 405–408.

[54] R. C. Jaeger and T. N. Blalock, *Microelectronic Circuit Design*, S. W. Director, Ed. McGraw-Hill, 2008.

Appendices

Appendix A

Numerical Model for Reverse Time Oscillator

```
%% Numerical Simulation for Exact Solvable Chaotic Oscillator
% Implements the systems described in
% <http://dx.doi.org/10.1063/1.3432557 A Matched Filter for Chaos> and
% <http://dx.doi.org/10.1063/1.4704813 Exact Folded-Band Chaotic
% Oscillator>.

%% Function Definition
% *oscillator* takes three inputs: _t_stop_ sets the stop time, _t_step_
% sets the solver step time, _band_
% controls which band the system is in (0 for folded, 1 for shift), _beta_
% is the system control parameter, and _u_0_ is the initial condition.
% Outputs are column vectors containing all solution values of t, u,
% du/dt, and s, respectively, as well as the time points and
% solution values of the overall solution's return points.
function [t_sol, u_sol, s_sol, du_sol, t_return_sol, u_return_sol] = ...
    oscillator_rt(t_stop, t_step, band, beta, u_0)

    clear t_n s sym u_m;

    if (band == 1)
        % calculate the map values
        u_m(1)=u_0;
        if (u_0 >= 0)
            s(1) = 1;
        else
            s(1) = -1;
        end

        t_n(1)=0;
        for t=2:t_stop+1
            if (u_m(t-1) >= 0)
                u_m(t)=(exp(beta)*u_m(t-1)-(exp(beta)-1));
                s(t) = 1;
            else
                u_m(t)=(exp(beta)*u_m(t-1)+(exp(beta)-1));
                s(t) = -1;
            end
            t_n(t)=t-1;
        end
    else
        u_m(1)=u_0;
        t_mx(1)=0;
```

```
j=1;
    if (u_0 > 0) && (u_0 <= 1)
        s(j)=0;
        t_n(j)=0;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
    elseif (u_0 > 1) && (u_0 < (1+exp(-beta/2)))
        s(j)=1;
        t_n(j)=0;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
    else
        s(j)=1;
        t_n(j)=0;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
    end
for t=2:t_stop+1
    if (u_m(t-1) > 0) && (u_m(t-1) <= 1)
        u_m(t)=(exp(beta)*u_m(t-1));
        t_mx(t)=t_n(j-1);
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
    elseif (u_m(t-1) > 1) && (u_m(t-1) < (1+exp(-beta/2)))
        u_m(t)=(-exp((3*beta)/2)*u_m(t-1)+(exp(beta)+exp((3*beta)/2)));
        t_mx(t)=t_n(j-1);
        s(j)=1;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
        s(j)=0;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
    else
        u_m(t)=(exp(beta)*u_m(t-1)-(exp(beta)+exp(beta/2)));
        t_mx(t)=t_n(j-1);
        s(j)=1;
        t_n(j)=t_n(j-1)+0.5;
        j=j+1;
```

```
97                  s(j)=0;
98                  t_n(j)=t_n(j-1)+0.5;
99                  j=j+1;
100             end
101         end
102     end
103
104     clear t;
105
106     %% Set Operating Parameters
107     omega = 2*pi;
108
109     % keep track of how many time units the ODE has been solved for
110     % must be initially 0
111     t_run = 0;
112
113     % initial start and stop times
114     t_s = 0;
115     t_f = 5;
116
117     % initial values for du_1/dt and du_2/dt
118     du_1dt_0 = 0;
119     du_2dt_0 = 0;
120     du_p = 0;
121     du_p_prev = 0;
122
123     % matrices to store final solution; initially empty
124     t_sol = [];
125     t_return_sol = 0;
126     u_sol = [];
127     u_return_sol = 0;
128     s_sol = [];
129
130     % tell the ode solver to evaluate the function events for event detection
131     options = odeset('Events',@events);
132
133     %% Main Loop
134     while (t_run < t_stop)
135         % call ode15s to solve the differential equation (see below)
136         %
137         % return values:
138         % t - vector of all time points where solution was calculated
139         % u - values for solution at corresponding time points
140         % te - time where event was detected
141         % ue - solution value at event
142         [t, u, te, ue] = ode15s(@(t,u) f(t, u, beta, omega, s, t_n, band), ...
143             [t_s:t_step:t_f], [du_1dt_0, du_2dt_0], options);
144
145         % append data from this run to the final solution matrices
146         % omit the final data point since it will be solved in the next
147         % iteration
148         t_sol = [t_sol; t(1:(length(t) - 1))];
149         u_sol = [u_sol; u(1:(length(u) - 1),:)];
150
```

```
151         % ue sometimes has more than one row; always select the last one
152         [m, n] = size(ue);
153         if (m > 1)
154             ue(1) = ue(length(ue), 1);
155             ue(2) = ue(length(ue), 2);
156         end
157 
158         % store the value of du at the point before the return to help
159         % determine its direction
160         du_p = u(length(u)-1, 2);
161 
162         % check that the sign of u and du is positive and that du isn't
163         % extremely small (this eliminates the point when s switches from
164         % high to low) to get the correct timing
165         if (band == 0)
166                       if ((sign(ue(1)) == 1) && (sign(du_p) == 1) ...
167                     && (abs(du_p) > 1e-5))
168                 % this event has detected a return, store it
169                 t_return_sol = [t_return_sol; max(t)];
170                 u_return_sol = [u_return_sol; ue(1)];
171             end
172         % for shift band, check if t=t+1 by subtracting the previous
173         % return time from the current time and then checking to make sure
174         % this isn't the zero crossing that occurs at t=t+0.5
175         else
176             if ((max(t)-t_return_sol(length(t_return_sol))) > 0.6)
177                 % this event has detected a return, store it
178                 t_return_sol = [t_return_sol; max(t)];
179                 u_return_sol = [u_return_sol; ue(1)];
180             end
181         end
182 
183         % update the run time with the maximum value of the t vector
184         t_run = max(t);
185 
186         % set new start time to point where event was triggered
187         t_s = te(length(te));
188         % set new stop time far away from new start time to insure ode15s
189         % runs long enough to find the next event
190         t_f = t_s + 5;
191         % set new initial conditions to solution value at event
192         du_1dt_0 = ue(1);
193         du_2dt_0 = ue(2);
194     end
195 
196     %% Format and Plot Results
197 
198     % split u_sol into du/dt and u
199     du_sol = u_sol(:, 2);
200     u_sol = u_sol(:, 1);
201 
202     if (band == 1)
203         for i=1:length(t_sol)
204             s_sol(i)=s(floor(t_sol(i)+1))*0.025;
```

```
205         end
206     else
207          for i=1:length(t_sol)
208             k = find(t_n==round(t_sol(i)));
209             s_sol(i)=s(k)*0.025;
210          end
211     end
212     % plot time series
213     figure(1)
214     hold on
215     plot(t_sol, u_sol);
216     plot(t_sol, du_sol, 'm');
217     plot(t_sol, s_sol, 'r');
218     % mark return locations
219     for i=1:(length(u_return_sol))
220         plot(t_return_sol(i), u_return_sol(i), 'g*');
221     end
222     xlabel('t');
223     ylabel('u(t), s(t)');
224     legend('u(t)', 's(t)', 'return');
225     hold off
226     % plot phase space
227     figure(2)
228     plot(u_sol, du_sol);
229     xlabel('u');
230     ylabel('du/dt');
231     % plot return map
232     figure(3)
233     hold on
234     for i=1:(length(u_return_sol)-1)
235         plot(u_return_sol(i+1), u_return_sol(i), '*');
236     end
237     xlabel('u(t)');
238     ylabel('u(t+1)');
239     hold off
240 end
241
242 %% *event* - Event Function for Detecting Zero Crossing of the Derivative
243 %
244 % *event* takes two inputs: the current values of _u_ and _t_ and returns
245 % these values in _value_ if du/dt = 0 is detected. _isterminal_
246 % returns 1 to stop integration and _direction_ returns if
247 % du/dt = 0 was approached from the positive or negative
248 % direction.
249 function [value, isterminal, direction] = events(t, u)
250     % detect du/dt = 0
251     value = u(2);
252     isterminal = 1;
253     % detect zero crossings from either direction
254     direction = 0;
255 end
256
257 %% *f* - Implements the Differential Equation
258 %
```

```
% *f* takes five inputs: the current valeues of _u_ and _t_, the system
% parameters _beta_ and _omega_, and the current value of the discrete
% state _s_. The output is a column vector containing the solved values for
% du1/dt and du2/dt.
%
% To determine du1/dt and du2/dt, the original
% second-order equation can be split into two first-order equations as
% follows:
%
% d^2u/du^2 - 2β du/dt + (ω^2 + β^2) * (u - s) = 0
%
% d^2u/du^2 = 2β du/dt - (ω^2 + β^2) * (u - s)
%
% Two new variables are defined as
% u1 = u and u2 = du/dt
%
% From these, it can be determined that
% du1/dt = u2 and
% du2/dt = 2βu2 - (ω^2 + β^2) * (u1 - s)
function dudt = f(t, u, beta, omega, s, t_n, band)
    dudt = zeros(2, 1);
    dudt(1) = u(2);
    dudt(2) = sgn(t, s, t_n, band)-2*beta*u(2)-((beta*beta+omega*omega)*(u(1)));
end

%% *sgn* - Implements the sgn Function
% *sgn* takes a single input _x_ and returns a value _y_ determined by:
%
% x ≥ 0 → y = 1; x < 0 → y = -1
function s_out = sgn(t, s, t_n, band)
    if (band == 1)
        s_out=s(floor(t)+1);
    else
        k = find(t_n==round(t));
        s_out=s(k);
    end
end


```

Appendix B
Routine for Determining Symbolic Dynamics of Skew Tent Map

```
function [sequences, init_cons] = ...
    discrete_chaos_codingfunc(beta, init_con_step, symbols)

    % initial condition index
    i=1;

    % initial condition loop
    % for each value of beta, the return map is calculated for the full
    % range of possible initial conditions at the entered step size
    for u_0=0:init_con_step:exp(beta)
        u(1)=u_0;

        % determine successive maxima based on equation
        % run for enough iterations to generate the number of symbols
        % entered
        for t=2:(symbols+1)
            if (u(t-1) > 0) && (u(t-1) <= 1)
                u(t)=(exp(beta)*u(t-1));
                s(t-1)='A';
            elseif (u(t-1) > 1) && (u(t-1) < (1+exp(-beta/2)))
                u(t)=(-exp((3*beta)/2)*u(t-1)+(exp(beta)+exp((3*beta)/2)));
                s(t-1)='B';
            else
                u(t)=(exp(beta)*u(t-1)-(exp(beta)+exp(beta/2)));
                s(t-1)='C';
            end
        end

        %generate bitstream
        j=1;
        for k=1:length(s);
            if s(k) == 'A'
                s_bits(j)='0';
                j=j+1;
                s_bits(j)='0';
                j=j+1;
            elseif s(k) == 'B'
                s_bits(j)='1';
                j=j+1;
                s_bits(j)='0';
                j=j+1;
                s_bits(j)='0';
                j=j+1;
            else
```

```
45                  s_bits(j)='1';
46                  j=j+1;
47                  s_bits(j)='0';
48                  j=j+1;
49              end
50          end
51
52          % convert the bits into a cell and store in the overall array
53          bits_itr(i)=cellstr(strcat(s_bits));
54          % store the initial condition for the corresponding bitstream
55          init_con_itr(i) = u_0;
56
57          % clear results of current run and iterate initial condition index
58          clear u;
59          clear s;
60          clear s_bits;
61          i=i+1;
62      end
63
64
65  figure(1)
66  hold on
67  % convert each bitstream into decimal form for plotting
68  for i=1:length(bits_itr)
69      s_1=bits_itr(i);
70      s_1=s_1(~cellfun(@isempty, s_1));
71      plot(init_con_itr(i), bin2dec(char(s_1)),  '*r');
72      bits_out(i)=bin2dec(char(s_1));
73  end
74  xlabel('u_0');
75  ylabel('Decimal Value of Bitstream');
76  hold off
77
78  % return the bitstreams and initial condition for each step
79  sequences = bits_itr;
80  init_cons = init_con_itr;
81  csvwrite('init_cons.csv', init_con_itr');
82  csvwrite('bits_out.csv', bits_out');
83
84  function [sequences, init_cons, lut] = ...
85      discrete_chaos_shift_codingfunc(beta, init_con_step, symbols)
86
87      % initial condition index
88      i=1;
89
90      % initial condition loop
91      % for each value of beta, the return map is calculated for the full
92      % range of possible initial conditions at the entered step size
93      for u_0=-(exp(beta)-1):init_con_step:(exp(beta)-1)
94          u(1)=u_0;
95
96          % determine successive maxima based on equation
97          % run for enough iterations to generate the number of symbols
98          % entered
```

```
        for t=2:(symbols+1)
            if (u(t-1) >= 0)
                u(t)=(exp(beta)*u(t-1)-(exp(beta)-1));
                s(t-1) = '1';
            else
                u(t)=(exp(beta)*u(t-1)+(exp(beta)-1));
                s(t-1) = '0';
            end
        end

        % convert the bits into a cell and store in the overall array
        bits_itr(i)=cellstr(strcat(s));
        % store the initial condition for the corresponding bitstream
        init_con_itr(i) = u_0;

        % clear results of current run and iterate initial condition index
        clear u;
        clear s;
        i=i+1;
    end


figure(1)
hold on
% convert each bitstream into decimal form for plotting
for i=1:length(bits_itr)
    s_1=bits_itr(i);
    s_1=s_1(~cellfun(@isempty, s_1));
    plot(init_con_itr(i), bin2dec(char(s_1)), '*r');
    bits_out(i)=bin2dec(char(s_1));
end
xlabel('u_0');
ylabel('Decimal Value of Bitstream');
hold off

% return the bitstreams and initial condition for each step
sequences = bits_itr;
init_cons = init_con_itr;
csvwrite('init_cons.csv', init_con_itr');
csvwrite('bits_out.csv', bits_out');
```

Appendix C
HDL Code for Shift Map

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
-- below are for fixed point math support
library IEEE_proposed;
use IEEE_proposed.fixed_float_types.all;
use IEEE_proposed.fixed_pkg.all;

entity oscillator_shift is port (
  clk: in std_logic; -- IN: clock
  en: in std_logic; -- IN: enable
  ctrl: in std_logic; -- IN: indicates that new initial condition is present
  u_0: in std_logic_vector (31 downto 0); -- IN: new initial condition
  u: out std_logic_vector (31 downto 0); -- OUT: system state
  s: out std_logic; -- OUT: bitstream
  ctrl_ack: out std_logic); -- OUT: indicates that new initial condition has been read
end oscillator_shift;

architecture arch of oscillator_shift is

type del_states is (d1, d2);

-- values for signed Q format numbers Qm.n
constant m: integer := 4; -- bits for integer portion (signed, two's complement)
constant n: integer := 28; -- bits for fractional portion

-- calculated for beta=0.99*ln(2)
constant c1: sfixed (m-1 downto -n) := to_sfixed(1.986184990874072, m-1, -n); --
 ↪  exp(beta)
constant c2: sfixed (m-1 downto -n) := to_sfixed(0.986184990874072, m-1, -n); --
 ↪  (exp(beta)-1)

signal n_delay: del_states := d2; -- keeps track of how many cycles to wait before next u
 ↪  update;
signal u_n: sfixed (m-1 downto -n) := to_sfixed(0.5, m-1, -n); -- operating copy of u,
 ↪  initially 0.5
signal s_n: std_logic := '0';
signal tmp: sfixed (m-1 downto -n) := to_sfixed(0, m-1, -n); -- memory for multi-step
 ↪  math operations, initially 0

begin
        process(clk, en)
        begin
```

```
40                 if (en = '0') then
41                         n_delay <= d2;
42                         u <= "00000000000000000000000000000000";
43                         s <= '0';
44                 elsif (rising_edge(clk)) then
45         case n_delay is
46           when d2 =>
47             n_delay <= d1;
48             u <= to_slv(u_n);
49             s <= s_n;
50             tmp <= resize(arg => c1*u_n, size_res => u_n); -- exp(beta)*u_n
51             ctrl_ack <= '0';
52           when d1 =>
53             n_delay <= d2;
54             if (ctrl = '1') then -- read in new initial condition if the ctrl flag is set
55               u_n <= to_sfixed(u_0, 3 ,-28);
56               ctrl_ack <= '1';
57             elsif (tmp >= to_sfixed(0, 3 ,-28)) then
58               u_n <= resize(arg => tmp-c2, size_res => u_n); -- exp(beta)*u_n -
                  ↪  (exp(beta)-1) (s_n = 1)
59               s_n <= '1';
60             else
61               u_n <= resize(arg => tmp+c2, size_res => u_n); -- exp(beta)*u_n +
                  ↪  (exp(beta)-1) (s_n = -1)
62               s_n <= '0';
63             end if;
64         end case;
65                  end if;
66     end process;
67 end arch;
```

Appendix D
HDL Code for Skew Tent Map

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_signed.all;
use IEEE.std_logic_arith.all;
-- below are for fixed point math support
library IEEE_proposed;
use IEEE_proposed.fixed_float_types.all;
use IEEE_proposed.fixed_pkg.all;

entity oscillator is port (
  clk: in std_logic; -- IN: clock
  en: in std_logic; -- IN: enable
  ctrl: in std_logic; -- IN: indicates that new initial condition is present
  u_0: in std_logic_vector (31 downto 0); -- IN: new initial condition
  u: out std_logic_vector (31 downto 0); -- OUT: system state
  s: out std_logic; -- OUT: bitstream
  ctrl_ack: out std_logic); -- OUT: indicates that new initial condition has been read
end oscillator;

architecture arch of oscillator is

type osc_states is (A, B, C);
type del_states is (d1, d2, d3, d4, d5, d6);

-- values for signed Q format numbers Qm.n
constant m: integer := 4; -- bits for integer portion (signed, two's complement)
constant n: integer := 28; -- bits for fractional portion

-- DAC resolution
constant dac_bits: integer := 14;

-- calculated for beta=0.81*ln(2)
--constant c1: sfixed (m-1 downto -n) := to_sfixed(1.753211442632070, m-1, -n); --
 ↪  exp(beta)
--constant c2: sfixed (m-1 downto -n) := to_sfixed(1.755236292781413, m-1, -n); --
 ↪  1+exp(-beta/2)
--constant c3: sfixed (m-1 downto -n) := to_sfixed(4.074619271399510, m-1, -n); --
 ↪  exp(beta)+exp(3*beta/2)
--constant c4: sfixed (m-1 downto -n) := to_sfixed(3.077300353027467, m-1, -n); --
 ↪  exp(beta)+exp(beta/2)
--constant c5: sfixed (m-1 downto -n) := to_sfixed(-2.321407828767440, m-1, -n); --
 ↪  -exp(3*beta/2)

-- calculated for beta=1.385*ln(2)
```

```
40 constant c1: sfixed (m-1 downto -n) := to_sfixed(2.611719574177835, m-1, -n); --
    ↪  exp(beta)
41 constant c2: sfixed (m-1 downto -n) := to_sfixed(1.618780655219275, m-1, -n); --
    ↪  1+exp(-beta/2)
42 constant c3: sfixed (m-1 downto -n) := to_sfixed(6.832471390105772, m-1, -n); --
    ↪  exp(beta)+exp(3*beta/2)
43 constant c4: sfixed (m-1 downto -n) := to_sfixed(4.227801123536603, m-1, -n); --
    ↪  exp(beta)+exp(beta/2)
44 constant c5: sfixed (m-1 downto -n) := to_sfixed(-4.220751815927937, m-1, -n); --
    ↪  -exp(3*beta/2)
45 constant u_dac_scale: sfixed (dac_bits-1 downto 0) := to_sfixed(4096, dac_bits-1, 0); --
    ↪  constant for scaling to DAC resolution
46
47 --signal c_delay: del_states := d6; -- keeps track of how many cycles to wait before next
    ↪  u update;
48 signal n_delay: del_states := d1; -- keeps track of how many cycles to wait before next u
    ↪  update;
49 signal state: osc_states := A; -- keeps track of if the system is in state A, B, or C
50 signal u_n: sfixed (m-1 downto -n) := to_sfixed(0.5, m-1, -n); -- operating copy of u,
    ↪  initially 0.5
51 signal tmp: sfixed (m-1 downto -n) := to_sfixed(0, m-1, -n); -- memory for multi-step
    ↪  math operations, initially 0
52 --signal u_dac_tmp: sfixed (dac_bits-1 downto 0) := to_sfixed(0, dac_bits-1, 0);
53
54 begin
55         process(clk, en)
56         begin
57                 if (en = '0') then
58                         n_delay <= d1;
59                         u <= "00000000000000000000000000000000";
60                 elsif (rising_edge(clk)) then
61         case n_delay is
62           when d6 =>
63             n_delay <= d5;
64             s <= '1'; -- first of two hi cycles for s in state B
65           when d5 =>
66             n_delay <= d4;
67             s <= '1'; -- second of two hi cycles for s in state B
68             u <= to_slv(u_n); -- convert u to std_logic_vector for output
69             -- multiply u by constant to convert fractional part to integer, then
                ↪  truncate to DAC resolution
70           when d4 =>
71             case state is
72               when C =>  -- case C: exp(beta)*u
73                 tmp <= resize(arg => c1*u_n, size_res => u_n);
74                 s <= '1'; -- first of two hi cycles for s in state C
75               when B => -- case B: -exp(3*beta/2)*u
76                 tmp <= resize(arg => c5*u_n, size_res => u_n);
77                 s <= '0'; -- first of four lo cycles for s in state B
78               when others => -- no operation for case A
79                 s <= '0'; -- first of four lo cycles for s in state A
80             end case;
81             n_delay <= d3;
82           when d3 =>
```

```
83              if (state /= B) then -- update u if this is the entry point (case A or C)
84                u <= to_slv(u_n); -- convert u to std_logic_vector for output
85                -- multiply u by constant to convert fractional part to integer, then
                   ↪  truncate to DAC resolution
86              end if;
87              n_delay <= d2;
88            when d2 =>
89              if (ctrl = '1') then -- read in new initial condition if the ctrl flag is set
90                u_n <= to_sfixed(u_0, 3 ,-28);
91                ctrl_ack <= '1'; -- set the ctrl_ack flag to acknowledge that u_0 has been
                   ↪  read
92                case state is
93                  when C => -- case C: exp(beta)*u - (exp(beta/2)+exp(beta))
94                    s <= '1'; -- second of two hi cycles for s in state C
95                  when B => -- case B: -exp(3*beta/2)*u + (exp(beta)+exp(3*beta/2))
96                    s <= '0'; -- second of four lo cycles for s in state B
97                  when A => -- case A: exp(beta)*u
98                    s <= '0'; -- second of four lo cycles for s in state A
99                  when others =>
100                   null;
101               end case;
102                                   else
103               case state is
104                 when C => -- case C: exp(beta)*u - (exp(beta/2)+exp(beta))
105                   u_n <= resize(arg => tmp-c4, size_res => tmp);
106                   s <= '1'; -- second of two hi cycles for s in state C
107                 when B => -- case B: -exp(3*beta/2)*u + (exp(beta)+exp(3*beta/2))
108                   u_n <= resize(arg => tmp+c3, size_res => tmp);
109                   s <= '0'; -- second of four lo cycles for s in state B
110                 when A => -- case A: exp(beta)*u
111                   u_n <= resize(arg => c1*u_n, size_res => u_n);
112                   s <= '0'; -- second of four lo cycles for s in state A
113                 when others =>
114                   null;
115               end case;
116             end if;
117             n_delay <= d1;
118             s <= '0'; -- third of four lo cycles for s in all states
119           when d1 =>
120             -- case A: u > 0 and u <= 1
121             if (u_n > to_sfixed(0, 3 ,-28) and u_n <= to_sfixed(1, 3 ,-28)) then
122               n_delay <= d4;
123               state <= A;
124             -- case B: u > 1 and u < 1+exp(-beta/2)
125             elsif (u_n > to_sfixed(1, 3 ,-28) and u_n < c2) then
126               n_delay <= d6;
127               state <= B;
128             -- case C: u > 1+exp(-beta/2)
129             else
130               n_delay <= d4;
131               state <= C;
132             end if;
133             s <= '0'; -- fourth of four lo cycles for s in all states
134             ctrl_ack <= '0'; -- reset the ctrl_ack flag
```

```
135          end case;
136                   end if;
137      end process;
138  end arch;
```