Design Document

FPGA Based Modem for Amateur Radio Satellite Communication

Submitted To:

Professor Dennis Silage
Senior Design Project I and II
Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

April 30, 2014

C. Destin, B. Keith, & B. Thibodeau

Programmable Communication Group

Faculty Advisor: Dr. Dennis Silage

Temple University
College of Engineering
1947 North 12th Street
Philadelphia, Pennsylvania 19122

For further information, please contact Dr. Dennis Silage (email: silage@temple.edu).

Team Members	Cedric Destin, Brandon Keith, Brian Thibodeau	
Advisor(s)	Dennis Silage, PhD	
Coordinator	Thomas Sullivan, PhD	
Department(s)	Electrical and Computer Engineering	
Project Title	FPGA Based Modem for Amateur Radio Satellite Communication	
Abstract	Amateur radio satellite telemetry is the process of using the amateur radio frequency bands to transmit telemetry data from a miniaturized low-Earth orbiting satellite to a ground station. The most prevalent means of transmitting telemetry data down to Earth is not nearly as power-efficient as it could be. Inefficient power usage makes amateur satellite telemetry an expensive and esoteric hobby to get involved with. As a result this senior design team aimed to demonstrate how concatenated forward error correction (FEC) codes can make amateur satellite telemetry more power-efficient, and hence make the hobby more accessible to prospective amateur satellite operators. Specifically, we wanted to use FPGA hardware to implement a BPSK modem with a (2,1,7) convolutional encoder and Viterbi decoder. However implementation issues with the Viterbi Decoder resulted in FEC being removed. The final product was a 1200 b/sec BPSK modem.	
URL	https://sites.google.com/a/temple.edu/programmable-communication-group/	

EXECUTIVE SUMMARY

The objective of this senior design project was to demonstrate that amateur radio satellite communications (SATCOM) equipment can be developed rather inexpensively using re-configurable digital hardware technology – the FPGA. Satellite communication equipment in the commercial and military sectors are primarily based on analog and digital integrated circuit topologies, yet the typical amateur radio satellite ground station is composed of equipment based on the obsolete discrete circuit topologies. One particular component of the amateur radio ground station, the modem, is often found in the discrete circuit format. This senior design project aimed to bridge the gap between amateur radio SATCOM and professional SATCOM by developing a modem using modern FPGA hardware platforms.

The biggest takeaway from this senior design project should be that the functionality of bulky communications equipment can be implemented on a single modern, inexpensive, and flexible hardware platform. Additionally, the FPGA hardware platform allows for rapid prototyping of designs. This rapid prototyping cannot be performed while developing discrete logic circuits. Not only does it take a lot of time to re-implement a new discrete logic circuit, but multiple digital logic circuit iterations can be rather expensive as well. The FPGA hardware platform allows for simulating circuits at various levels of reality before implementing the final rendering on hardware. This allows designs to be prototyped at a much faster pace than what is possible using discrete logic circuits. FPGA technology has advanced enough to the point that a curious amateur radio operator could learn to implement a new piece of communication equipment in a short period of time using this FPGA hardware platform.

This senior design project employed a top-down design approach that could be replicated for free by any amateur radio operator curious to dig into the details of designing miniature satellite communications equipment. We began the project by researching various miniature satellite modem designs. We decided to implement a cost-efficient and power-efficient design. This design can be implemented using a low-cost FPGA hardware platform and free software development tools and intellectual property. After the research and design selection, we used a free software tool (MATLAB) to simulate the design at a high-level for system performance assessment. We then acquainted ourselves with the inexpensive FPGA hardware platform and then began to implement the simulated design onto the platform. The wide suite of free intellectual property allows for implementing complex designs at a miniscule cost. We used relatively inexpensive hardware and software tools for design verification.

The content within this senior design document serves to provide a concrete example of satellite communications equipment being designed and implemented at a hobby-level cost – less than $500. This senior design team patiently awaits the day where amateur radio operators are tinkering and hacking away at miniature satellite communications in a way that is accessible, cost-effective, and future-proof. This senior design project is simply urging the day to arrive a little sooner.

Table of Contents

1. PROBLEM

1.1. Overall Objectives

It has been shown that forward error correction can dramatically improve bit error rate performance (BER) in amateur packet radio satellite telemetry links (Hsiao, et. al, 2000). Additionally, it has been shown that binary phase shift-keying (BPSK) modulation is more reliable and bandwidth-efficient than audio frequency shift keying (AFSK) modulation (Hsiao, et. al, 2000). A quick survey of active amateur radio satellites reveals that many of them use FSK modulation and lack forward error correction capabilities. This senior design project aims to demonstrate how BPSK modulation with forward error correction (specifically convolutional coding) makes amateur radio satellite telemetry more accessible and reliable. Consequently, this senior design project advocates for improved robustness in amateur packet radio communication systems, specifically in those systems dealing with satellite telemetry.

Amateur packet radio satellite telemetry is often unidirectional and does not benefit from automatic repeat request (ARQ) like in other bidirectional amateur packet radio communications (Hsiao, et. al, 2000). In other words, if even one bit of an AX.25 telemetry packet is received in error, the entire packet is discarded and cannot be re-transmitted (Karn, 1994). This means that beacon signals from the amateur satellites must be transmitted with enough power to ensure that the embedded telemetry packet is received without error (de Milliano, et. al, 2010). BPSK modulation with convolutional coding outperforms AFSK modulation in terms of BER performance. This fact could benefit amateur radio satellite telemetry by improving network reliability and accessibility. The enhanced network reliability could lower overall power consumption in amateur telemetry satellites (de Milliano, et. al, 2010), resulting in two benefits: 1) reduced cost of satellite construction, and 2) making amateur telemetry satellites more technologically and financially accessible to amateur satellite operators by reducing the size, cost, and complexity of ground station antennas (Karn, 2011).

The ultimate goal of this senior design project is to demonstrate the improved network reliability (in terms of BER) and accessibility (in terms of link margin) that results from implementing BPSK modulation with convolutional coding in amateur packet radio telemetry satellites and ground stations.

1.2. Historical and Economic Perspective

The standard digital modulation scheme used for amateur radio very-high frequency (VHF) and ultra-high frequency (UHF) operation is Bell 202 (Capitaine, et. al, 2010). Bell 202 provides AFSK modulation using 1200 Hz and 2200 Hz tones, with a resulting data rate of 1200 b/sec. It is typically used in the physical layer of the AX.25 data link layer protocol and this has been the case since the early 1980s (Karn, 1994). In 1984, when Bell 202 was a fairly new standard in the amateur radio community, Steve Goode, K9NG, performed an exhaustive bit error rate (BER) performance analysis of a standard Bell 202 modem (Goode, 1984). Goode found that 25 dB SNR at his FM receiver was necessary for high communication reliability. In other words, 25 dB SNR or higher was required to accurately receive 98% of incoming packets, which corresponded to a BER of 1.6e-5. Ralph Wallio, WORPK, figured out that with this BER, there is only a 1.603% chance of accurately receiving 117 consecutive 256-byte AX.25 packets (Wallio). Wallio concluded that “this is as Goode as it gets” and it is virtually impossible to get better results without error correction.

This poor reliability performance is not exclusive to amateur radio terrestrial communications. In 1995, it was demonstrated that error detection (not error correction) alone is not robust enough for amateur radio microsatellite communications (Hsiao, et. al, 2000). Particularly in unidirectional satellite communications, the harsh environmental conditions coupled with the microsatellite’s characteristically

low transmitter power make for very unreliable telemetry data links (Hsiao, et. al, 2000). It has been demonstrated that forward error correction, specifically convolutional coding, can generally correct up to 75 percent of errors (Hsiao, et. al, 2000). It was also demonstrated that 1200 b/sec BPSK provides much more reliable transmission quality than 1200 b/sec AFSK, irrespective to whether the VHF or UHF amateur bands are used. Moreover, it was demonstrated that BPSK occupies a considerably smaller frequency bandwidth than AFSK while possessing excellent anti-interference properties. And with a general tenfold BER performance increase for both 1200 b/sec AFSK and BPSK over 144 MHz VHF, implementing forward error correction for amateur satellite telemetry was clearly demonstrated to be better than not implementing forward error correction.

In 2003, the AAU-Cubesat was one of the first pico-satellites to be launched into space. Moreover, the miniaturized satellite harbored a communication subsystem that implemented both forward error correction and interleaving over 9600 b/sec Gaussian minimum shift-keying (GMSK) AX.25 (Alminde, et. al, 2002). The enhanced robustness and data rate was justified by the fact that it had to transmit approximately 1461 kilobytes (kB) of telemetry and picture data per day. This simply would not have been possible had the satellite not utilized error correction. However, it operated at 437.9 MHz, which meant that a 2-meter radio could not receive its telemetry data. This would particularly bother Phil Karn, KA9Q, who is a strong proponent of making robust satellite telemetry links accessible to the average amateur radio operator (Karn, 2011). Karn asserts that robust telemetry links (using forward error correction) reduce the cost of satellite construction and simplify ground antennas, making amateur radio satellite telemetry much more technologically and financially accessible to amateur satellite operators (Karn, 2011).

As amateur satellite designers foresee the next generation of miniature satellites (de Milliano, et. al, 2010), and as the next generation of amateur satellites equipped with robust communication schemes continue to ascend into space, and as miniature satellites become increasingly more financially and technologically accessible to amateur satellite operators, it must be clearly demonstrated to the amateur radio community how these advancements trump the ubiquitous 1200 b/sec AFSK AX.25. Hence, to reiterate, this senior design project hopes to clearly demonstrate the performance advantages that yield from using BPSK modulation with forward error correction in amateur satellite telemetry.

1.3. Candidate Solutions

Basic functions of a telemetry modem include baseband modulation/demodulation (line codes), passband modulation/demodulation, synchronization, and forward error correction. In the last century, many solutions have been proposed that trade performance in terms of bandwidth, transmission power and complexity. In this section we consider the following solutions:

1. Two forward error correcting codes - block codes and convolutional codes.
2. Two line codes – Non-Return to Zero (NRZ) and Differential code.
3. Two carrier recovery circuits – Costas Loop and squaring loop.
4. Two timing and data recovery circuits – open loop and closed loop (Early-Late gate).

The subsequent sections will consider the costs and benefits of the aforementioned solutions.

1.3.1. Forward Error Correction: Block and Convolutional Codes

Forward error correction (FEC) is a form of robust channel coding. It is used to correct errors that are injected into a digital communication link across a noisy propagation medium (channel). FEC codes fall into two general categories: block codes and convolutional codes.

NOTE: *At the time of writing this document, the Xilinx CORE Generator in Project Navigator ISE 14.6*

only consists of one block coder/decoder pair and one convolutional coder/decoder pair. The block coding pair consists of a Reed-Solomon coder and decoder. The convolutional coding pair consists of a convolutional encoder and a Viterbi decoder.

Deep space and satellite communication links are riddled with random errors across a very wide bandwidth (Nguyen, et. al, 2009). In addition to random errors in the satellite link, bursts of noise can corrupt an entire segment of a link resulting in burst errors (Murphy, et. al, 1994). Block codes are better suited for correcting burst errors while convolutional codes are better suited for correcting random errors (Viswanathan, 2013). A combination of block codes and convolutional codes, namely concatenating codes, are used in many systems to provide robustness against both kinds of errors (see Figure 1). This concatenating code consists of a coding chain and a decoding chain. The coding chain resides in the transmitter and consists of a Reed-Solomon encoder, followed by a block interleaver, then a convolutional encoder. The decoding chain resides in the receiver and undoes what the coding chain did. Namely, the decoding chain consists of a Viterbi (convolutional) decoder, followed by a block de-interleaver, then a Reed-Solomon decoder. This is illustrated in Figure 1.

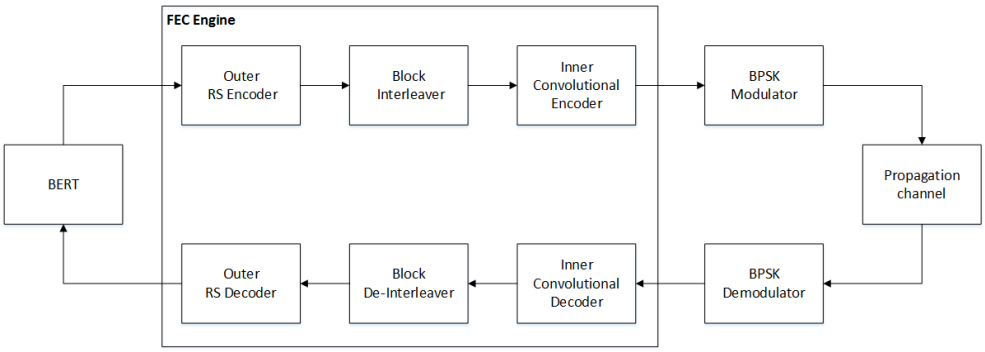

Figure 1. Simplified top-level diagram for a concatenating code scheme consisting of a block code pair, an interleaving pair, and a convolutional code pair.

The propagation medium for space communications is modeled well by the AWGN channel (Viswanathan, 2013). Furthermore, it is understood that AWGN provides maximum bit corruption and compared to other channel models, systems that perform best in AWGN also perform the best in real-life applications (Viswanathan, 2013). Hence, this senior design project will rely solely on the AWGN channel to represent our propagation medium.

Figure 2 shows the propagation medium being modeled by the AWGN channel. The AWGN channel is a random noise channel, not a noise channel with burst errors. Being that convolutional coding excels at correcting random errors, it is logical that convolutional coding alone pairs well with the AWGN channel. Therefore, convolutional coding is the only forward error correction scheme used in this senior design project (shown in Figure 2).

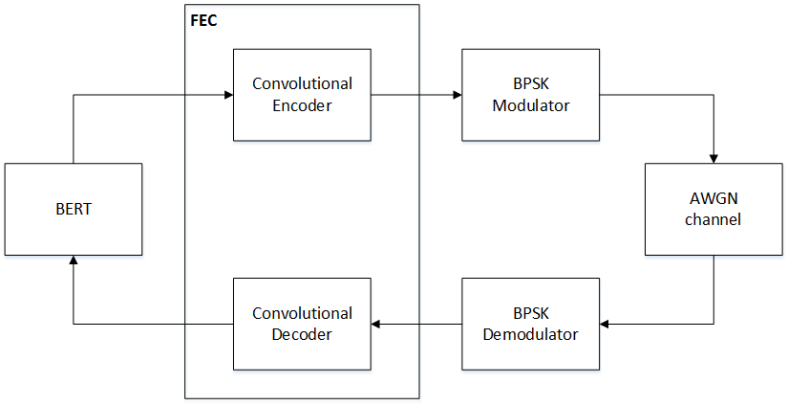

Figure 2. Simplified top-level diagram for a BPSK modem using a convolutional coding scheme for forward error correction. An AWGN channel is used as the propagation medium due its ability to model communication links. Convolutional coding is used to correct random errors that result due to the random noise of the AWGN channel.

1.3.2. Line Coding: Non Return Zero and Manchester

There are three criteria used for evaluating the performance of line codes: interference and noise immunity, bandwidth, and synchronization capabilities. These criteria are used for determining the appropriate line code that should be used in modems. NRZ is the most common line code as it appears naturally in digital logic. A binary one (b_1) is represented by a positive voltage while a binary zero (b_0) is represented by zero voltage. Conversely, Manchester line code represents a 'b_1' by a transition from zero volts to a positive voltage during the second half of the bit period while a 'b_0' is encoded as a transition from a positive voltage to zero volts during the first half of the bit period. Compared to NRZ, this means that Manchester code has two level transitions during one bit period while NRZ has only one. This presents a tradeoff between NRZ and Manchester in terms of synchronization and bandwidth. The two level transitions during each bit period means that the receiver can easily extract the transmitted clock to use for synchronization between the transmitter and receiver. The tradeoff is an increase in bandwidth due to the more frequent transitions, but how do these line codes perform in a noisy environment? The simple answer is that they perform the same. It can be shown that the energy, E, in NRZ and Manchester is V^2T_b, where T_b is bit period. It can also be shown that each line code has a theoretical probability of bit error of:

$$\boldsymbol{P_b = Q\left(\sqrt{\frac{2E}{N_o}}\right)}, \tag{1}$$

Since NRZ and Manchester share the same theoretical probability of bit error and contain the same energy, they both perform equally likely in a noisy environment. Thus the decision for choosing NRZ or Manchester resides in what is more important, bandwidth or synchronization? In our design, we decided that bandwidth was more important and thus NRZ was chosen over Manchester code.

1.3.3. Carrier Recovery: Squaring Loop and Costas Loop

The modem's demodulator is responsible for providing either coherent or non-coherent demodulation. Coherent demodulators require phase synchronization between the received signal and the locally generated oscillator. Conversely, Non-coherent demodulation does not require synchronization and

makes no attempt to estimate the phase of the received signal. The advantage of non-coherent modulation is that it does not require additional hardware like phase-locked loops which are used to lock onto the incoming carriers phase (Feigin, 2002). However, the AMSAT's we are interested in communicating with use BPSK for downlink and thus requires the design of a coherent demodulator.

The successful extraction of information from a received signal in a coherent demodulator requires both carrier and timing synchronization. Figure 3 illustrates the architecture of a typical coherent demodulator.

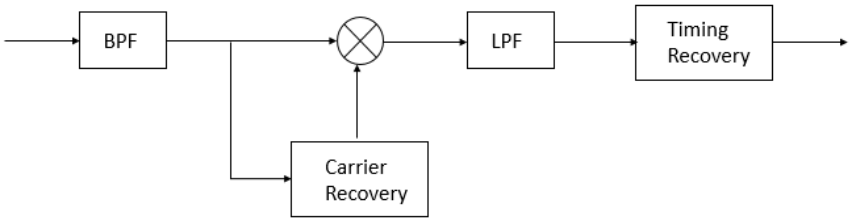

Figure 3. Received waveform takes two paths. First path extracts carrier for coherent demodulation and the second path recovers timing information. This architecture is based on the optimum binary receiver

The received signal from the transceiver is first processed by a band-pass filter to remove as much noise as possible and then sent to the carrier recovery circuit. Recovering the carrier is done in one of two ways, the squaring loop or the Costas loop. Each method utilizes phase-lock concepts and has its own advantages and disadvantages in terms of complexity and performance.

Carrier Recovery using Squaring Loop

The squaring loop is a popular choice for coherent demodulation of BPSK waveforms because it is mathematically easy to analyze and its hardware implementation is not as complex as the Costas loop. The squaring loop takes advantage of the fact that when the BPSK signal is squared, the phase offsets are removed leaving only a spectral component at twice the carrier frequency. A bandpass filter isolates the spectral component while also serving to remove any extraneous noise products resulting from the squaring. Following the band-pass filter, the signal is fed to a phase-lock loop (PLL) for phase and frequency tracking. Once the output of the voltage controlled oscillator (VCO) is locked in phase and frequency with the received signal, its frequency is divided by two. The resulting carrier is fed back to the correlator where it is mixed with the received BPSK waveform and the timing can be recovered (Nguyen & Shwedyk, 2009). The operation of the squaring loop is shown in Figure 4.

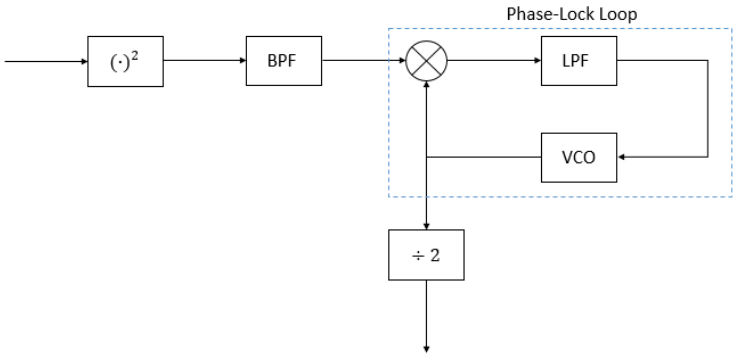

Figure 4. Squaring loop used for carrier recovery in the coherent demodulator. The Phase-Lock Loop utilizes feedback to track and lock onto in the received waveforms suppressed carrier

Carrier Recovery using Costas Loop

The second method for carrier recovery is the Costas Loop. Unlike the squaring loop whose only purpose is suppressed carrier reconstruction, the Costas loop is capable of synchronous data detection in addition to suppressed carrier reconstruction (Feigin, 2002). The received BPSK signal takes two paths in the Costas loop, the in-phase loop (top of Figure 4) and the quadrature loop (bottom of Figure 4). Unlike the PLL in the squaring loop, the Costas PLL uses an orthogonal carrier in the quadrature loop which allows direct demodulation of the BPSK signal without a need for squaring. The demodulated symbols are extracted from the in-phase arm once the loop is locked. This is illustrated in Figure 5.

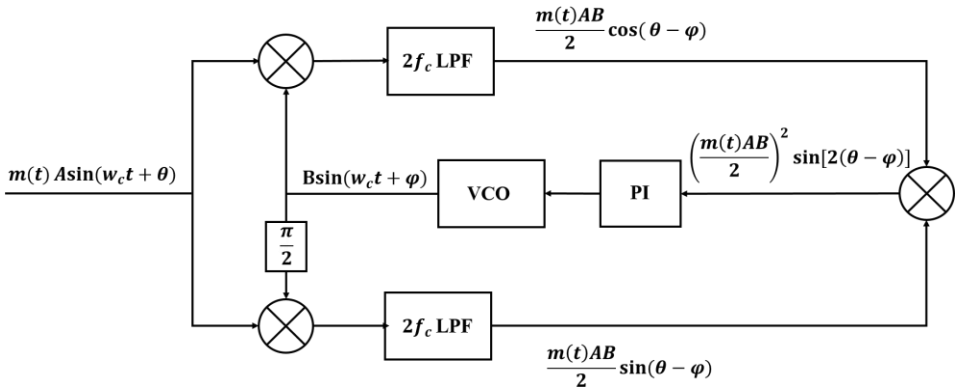

Figure 5. Costas loop used for suppressed carrier reconstruction as well as synchronous data detection.

Following a thorough analytical and experimental analysis of both the Costas loop and squaring loop in Simulink, it was decided that the Costas loop would be implemented for carrier recovery. The squaring loop demonstrated poor BER results as compared to the Costas Loop and the PLL proved unreliable to frequency steps and random 180 degree shifts. Alternatively, the Costas Loop demonstrated superior BER results and more reliable and robust carrier tracking to both phase and frequency steps. One of the downsides of the Costas loop is implementation of the arm filters. If these filters are not perfectly matched, then the loop's performance is degraded. However, high speed digital circuits like FPGA's

allow the design and implementation of identical filters thus alleviating the problem.

1.3.4. Clock and Data Recovery: Open Loop and Closed Loop Circuits

Timing recovery is the process of extracting a clock from the received signal so that the correct symbol determination can be made. The reason for this can be understood by recognizing that the local clock at the receiver is not synchronized with the transmitter clock and does not know when to sample the received data in order to make the correct symbol determination. In this section we consider two non-data aided architectures used for timing recovery. The first is an open loop architecture which is shown in Figure 6 and the second is a closed loop architecture shown in Figure 6. The closed loop circuit under consideration is also known as the Early-Late Gate.

In both methods it is assumed that the received signal is baseband and contains no spectral component. Thus the problem is similar to carrier phase recovery in coherent demodulators. In the open loop method, a spectral component is created by delaying the received signal by one half a bit time and then multiplying it with the original received signal. The result of the multiplication produces a spectral component at a rate of $1/T_b$ Hz. Then a simple band pass filter isolates the desired spectral component. Although simple to implement, the problem with the open loop method is that there is a non-zero tracking error that reduces system performance (Nguyen & Shwedyk, 2009).

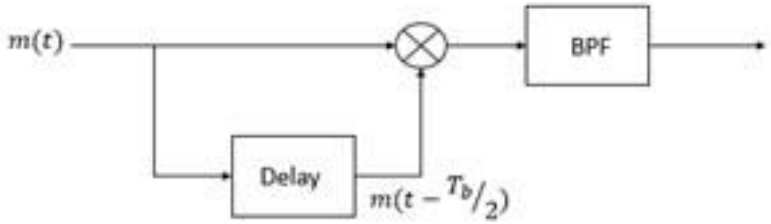

Figure 6. Open loop clock recovery circuit that produces a spectral component at $1/T_b$.

The non-zero tracking error and be eliminated by using a closed loop timing circuit instead. The Early-Late gate is one closed loop circuit that requires three samples during each bit period. If *m(t)* is the received baseband signal from the correlator, then the early gate integrates and samples *m(t)* early while the late gate integrates and samples *m(t)* late. The absolute value of the early and late samples are then compared to generate an error. The error signal drives a VCO which advances or retards the clock until the error is zero (Judd, 1996). When the error is zero, the output clock from the VCO is used to sample the received signal at the optimal time needed for correct symbol determination. Since the Early-Late gate synchronizer results in zero error, this method was chosen for timing recovery in our modems. A more detailed analysis of its operation is discussed in the Section 3, Approach.

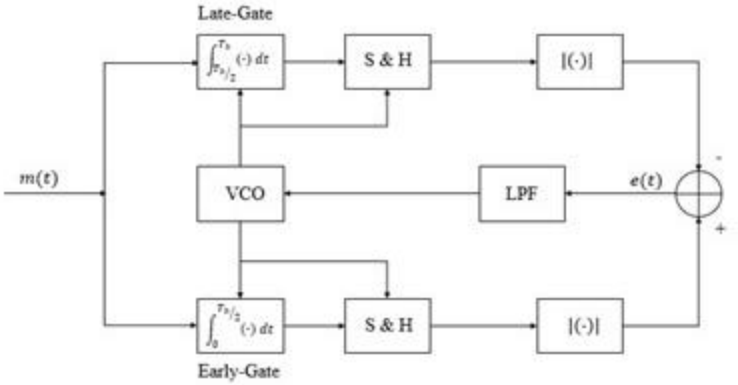

Figure 7. Architecture for the open- loop timing recovery (top) and the closed- loop (bottom)

1.4. Proposed Solution Concept

This senior design project will develop a software simulation (Simulink) and a hardware implementation (FPGA) of a 1200 b/sec FEC-BPSK modem (using soft-decision) with a (2, 1, 7) convolutional code. Synchronization in the demodulator is accomplished using the Costas Loop carry recovery circuit and the Early-late gate timing and data recovery circuit. We will gather measured BER performance data from the software simulation and the hardware implementation and compare it to the theoretical BER performance. When the measured and theoretical BER performances match closely, the software simulations and hardware implementations will be considered complete. The FPGA hardware implementation is expected to be the most important product of this senior design project.

1.5. Major Design and Implementation Challenges

The BER performance analysis of our FEC-BPSK modem is dependent upon the accessibility of an AWGN channel. Although Simulink provides an AWGN channel for the software BER analysis, Xilinx does not currently support an AWGN IP core to be used for our hardware analysis. Xilinx has discontinued their AWGN Core v1.0 since the release of ISE Design Suite 11 (Xilinx, 2009). Consequently, we had to find and use an AWGN core outside of the Xilinx product line.

The Costas loop carrier recovery circuit is expected to track and lock onto the transmitted carrier frequency under severe SNR conditions. Thus a major challenge is to design a robust Costas Loop that is capable of tracking changes to both phase and frequency offsets. This is traditionally accomplished using feedback control system concepts, but the Costas Loop (along with the conventional PLL) exhibit highly nonlinear behavior that make modeling a difficult task. However the Costas Loop can be linearized in terms of the phase using small angle approximations. It is important to note that the loop's behavior to large phase and frequency offsets requires extensive simulation to ensure the desired response is achieved and stability is maintained. Similar design challenges exist for the Early-Late gate circuit as it is also a feedback control system.

The transition from floating-point arithmetic used in software simulation and hardware implementation may prove to be troublesome. We understand that there is the **float** data type available in Verilog, but are design requires only fixed point implementation. Thus proper care must be taken to appropriately scale gains and filter coefficients so that the behavior of the fixed point hardware implementation still matches

the floating point software simulation.

1.6. Implications of Project Success

It was hinted in Section 1.1 (Overall Objective) and Section 1.2 (Historical Perspective) that this senior design team has identified a problem within the amateur radio community. According to amateur radio operator Jeff Davis, KE9V, amateur radio has somewhat of a *lost future* (Davis, 2010). In the earlier half of the 20th century, amateur radio operators led the forefront of "discovery and experimentation" in the industries of electronics and communications. This was the case because many amateur radio operators were in fact professional electronics technicians and electronics engineers that designed and implemented the next wave of commercial and military communications. Oftentimes, the budding amateur radio operator, a *neophyte* if you will, would go on to become the next electronics repairman or electronics engineer. However, Davis highlights the fact that at some point in the past, the amateur radio community reached somewhat of a crossroads. Up to that point in time, the amateur radio community had pioneered Frequency Modulation (FM) communications over ultra-high frequency (UHF) and very-high frequency (VHF) bands, stationed repeaters throughout the land for long-distance over-air communications, and launched amateur radio satellites into the heavens which led to improved methods for space communications in addition to low-cost spacecraft manufacturing and launch. Davis highlights the fact that although the non-amateur world would go on to produce cellular technology, drastically improved over-air communications, and intelligent military digital communications, the amateur radio community as a whole decided to dwell in the past as the future marched ahead without it.

This senior design team identified one amateur radio operator and notable electrical engineer, Dr. Phil Karn, KA9Q, in his efforts to secure the future of amateur radio. Like Jeff Davis, Dr. Karn is also aware of amateur radio's *lost future*. In a modem design article (Karn, 2011), Dr. Karn hints that making amateur radio communications more accessible to prospective amateur radio operators is one solution for securing the future of amateur radio. Specifically, in the design article, Karn identifies the fact that amateur radio satellite communications is mostly inaccessible to amateur radio operators because the equipment involved is too expensive and esoteric. Karn's philosophy is that by making amateur radio satellite communication accessible to all amateur radio operators, school demonstrations will be more commonplace and consequently more kids will want to become amateur radio operators. It is implied that if more kids become amateur radio operators, or *hams*, amateur radio in general cannot have a *lost future*.

Hence, according to Phil Karn, one solution to securing the future of amateur radio is to make amateur radio satellite communications more accessible to kids. In order to make amateur radio satellite communications more accessible to kids, the amateur radio equipment involved in said communications must be less expensive and esoteric. By expensive and esoteric, Karn is referring to state-of-the-art software-defined radio systems and bulky antennas. This kind of equipment is regarded as being too inaccessible for the typical school demonstration of amateur radio satellite communications. Instead, Karn emphasizes the fact that a standard 2-meter single sideband (SSB) transceiver and an inexpensive antenna system should be all that is required at these school demonstrations. Satellite communications in general requires relatively high-powered transmission of signals to overcome the high fading (energy loss) that results from electromagnetic waves propagating through space (Sklar, 2001). In fact, free space attenuates an electromagnetic wave more than any other form of power attenuation along a satellite communication link. Hence, it is often the case that transmitted signals between amateur packet radio satellites and ground stations either deal with high transmission power to acquire a digital communication link with high data reliability or lower transmission power and low data reliability and link efficiency. It is understood that if you increase the reliability (BER) of a communication link, you can communicate with less capable ground stations (de Milliano, et. al, 2010). "Less capable" ground stations includes stations with small, portable SMA antennas and little USB modem dongles (e.g. FunCube dongle). Consequently, amateur radio satellite communications could become more *accessible* to prospective amateur satellite

operators.

In a similar fashion as Phil Karn, KA9Q, and others (Hsiao, 2000), this senior design project aims to demonstrate that there are much more reliable digital communication schemes than are currently employed in most amateur radio satellites today. The intention of this senior design project is to provide concrete evidence that BPSK modulation with forward error-correcting codes can make amateur radio satellite communications more reliable and hence, more *accessible* to prospective amateur satellite operators. Perhaps budget link analyses of popular and prospective communication schemes, like showcased in this senior design project, would further persuade an amateur satellite designer to employ more reliable communication schemes in the ascending fleet of miniaturized amateur radio satellites.

2. DESIGN REQUIREMENTS

As discussed in the section 1.4 (Proposed Solution Concept), our analysis will require the design and implementation of a 1200 b/sec FEC-BPSK modem with a (2, 1, 7) convolutional code. The functional requirements of this system are summarized in Table 1 in section 2.1 followed by a summary of each. Non-functional requirements are listed in Table 2 in section 2.2.

2.1. Functional Design Constraints

Name	Description
Data Rate	The convolutional encoder is required to receive user data at a rate of 1200 b/sec. The convolutional encoder is required to transmit FEC data at a rate 2 x 1200 = 2400 b/sec. The BPSK modulator is required to receive serialized FEC data at a rate of 0.5 x 2400 b/sec = 1200 b/sec. The BPSK demodulator is required to deliver 3-bit soft-decision data at a rate of 3 x 2400 b/sec = 7200 b/sec. The Viterbi decoder is required to receive 3-bit soft-decision at a rate of 3 x 1200 = 3600 b/sec. The Viterbi decoder is required to deliver user data at a rate of 1200 b/sec.
Symbol Rate	The BPSK modulator is required to operate at 2400 symbols/sec. The BPSK demodulator is required to operate at 2400 symbols/sec.
Modulation/ Demodulation	BPSK modulation is demodulation is done coherently. 3-bit soft-decision encoding is implemented in the output of the demodulator.
Lock Time/range	It is required that the Costas Loop carrier recovery circuit track and lock onto the received carrier in less than 5 ms ($\approx$ 5 bit times). The Costas Loop should also be resilient to both phase and frequency offsets with a lock range of $\pm$200 Hz from the center frequency.
Operating Frequencies	The BPSK modems will modulate the data using a 4800 Hz carrier.
BER Performance (Eb/N0 dB) (over AWGN)	0.5 @ 0 dB 0.0783 @ 1 dB 0.0071 @ 2 dB 4.28e-4 @ 3 dB 1.74e-5 @ 4 dB 4.40e-7 @ 5 dB
Forward Error Correction	This modem will use the (2, 1, 7) convolutional code. Viterbi decoding will operate in 3-bit (fixed point) soft-decision decoding mode.

Interface(s)	Modified 19-bit AWGN core (Verilog) BERT: RS-232 (1200 b/sec, 1 parity bit, 1 stop bit, no flow control)

Table 1. Functional design constraints for the all three systems.

2.2. Non-Functional Design Constraints

These non-functional design constraints are based off of the Trenz Electronics Micromodule Spartan-6 XCSLX45-2CSG484I FPGA development board (product#: TE0630-00I) and Trenz Electronics Demo Carrier Board for Industrial Micromodule TE0300/TE0630 (product#: TE0304-00).

Type	Name	Description
Economic	Cost	FPGA Board: $204.64, Carrier Board: $81.03
Environmental	Temperature	FPGA Industrial grade: -40° C to 100° C
Environmental	Power Consumption	Carrier Board: USB bus power supply
Manufacturability	Dimensions	Carrier Board: 115 x 79 mm
Manufacturability	Weight	< 1 lb.

Table 2. Non-functional design constraints for test board.

3. APPROACH

3.1. Software Simulation Using Matlab/Simulink

Simulink provides a graphical design environment for rapid prototyping and simulation of the subsystems and circuits required by the FEC-BPSK modem. Each system, subsystem, and circuit was designed using a black box approach. Not only does this allow seamless integration of our individual designs, but it also helps organize the design modularly when transitioning to FPGA. Furthermore, Simulink provides the tools and block sets necessary to evaluate the performance of our system under a variety of conditions. Of particular importance is the evaluation of BER performance in AWGN and phase and frequency shifts. These are conditions that all modems are expected to able to handle.

The remainder of the section is organized as follows: section 3.1.1 will consider the transmitter-side of the forward error correction scheme. That is, the (2, 1, 7) convolutional encoder and required 2-bit serializer will be discussed. In section 3.1.2, the modulator subsystem is discussed. Then in 3.1.3, the demodulation subsystem of BPSK is discussed by examining the Costas Loop carrier recovery circuit and the Early-Late gate clock and data recovery circuit. The Simulink design if each circuit will be discussed in section 3.1.3. Lastly, section 3.1.4 will conclude the Software simulation section by examining soft-decision Viterbi decoding and how it can be used to reduce the BER of the BPSK modem.

3.1.1. Forward Error Correction: Convolutional Encoder & 2-bit Serializer

The convolutional encoder holds *k* bits of input data (*k*-bit message word) to generate an *n*-bit output. A constraint length *K* represents how many *k-bit* message words are used to process the *n-bit* output of the convolutional encoder. This would be regarded as a (*n*, *k*, *K*) convolutional encoder. The convolutional encoder is composed of a shift register with *K k-bit* stages and *n* modulo-2 adders. As an example, Figure 8 shows a (2, 1, 7) convolutional encoder. In fact, the (2, 1, 7) convolutional code was to be used solely throughout this senior design project. The (2, 1, 7) convolutional encoder comprises $K = 7$ stages (or, $K - 1 = 6$ *k*-bit delays) in its shift register and $n = 2$ modulo-2 adders. Each stage of the shift register holds $k =$

1 bits. The *code rate* of a convolutional encoder is *k/n*, so the code rate for this encoder is ½.

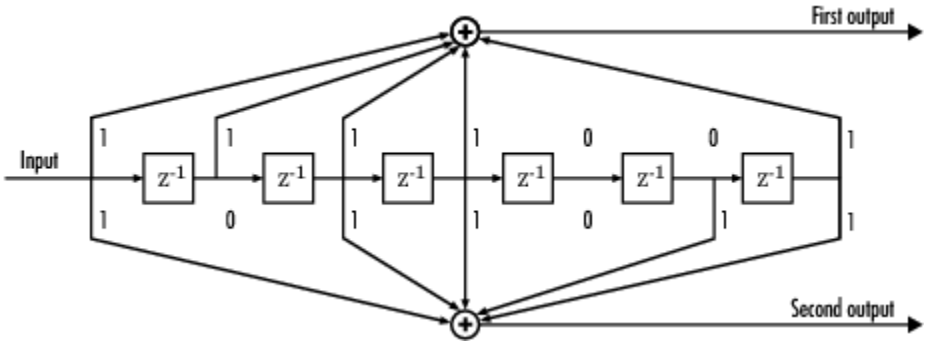

Figure 8. (Courtesy of Matlab®) A convolutional encoder (1/2 code rate, constraint length 7).

At each unit of time, *k* bits will shift to the next stage in the shift register, and *k* bits will shift into the first stage of the shift register. There are *K* stages for a group of *k* bits to shift into before it eventually shifts out of the shift register. At each unit of time, each of the *n* modulo-2 adders are sampled and these *n* bits are outputted by the convolutional encoder. In Figure 8, the top modulo-2 adder provides the first output bit and the bottom modulo-2 adder provides the second output. These two bits combined are the output of this convolutional encoder. Hence, one bit shifts into the encoder and two bits are produced by the encoder. The connections between the shift register stages and the modulo-2 adders, or the *generation matrices*, characterize the convolutional encoder. In other words, some permutations of connections have better error-correcting capabilities than other permutations of connections (Sklar, 2001).

The *trellis diagram* is widely used for showing the possible outputs of a convolutional encoder. However, in the case of the (2, 1, 7) convolutional encoder, the trellis diagram would compose 64 ($2^{K-1} = 2^{7-1}$) states. Consequently, it would not be practical or possible to display the full trellis diagram in this report. However, as a basic example, let's see how the convolutional encoder (shown in Figure 41) would process the following:

In the convolutional encoder: 1101000_2

1st output: $(1_2 + 1_2 + 0_2 + 1_2 + 0_2) \% 2 = 1_2$

2nd output: $(1_2 + 0_2 + 1_2 + 0_2 + 0_2) \% 2 = 0_2$

2-bit output: 01_2

The *poly2trellis* function is used by Simulink to generate the functionality of the *Convolutional Encoder*. Hence, the command *poly2trellis(7, [171 133])* was entered in the *trellis structure* parameter field. The argument is interpreted as a convolutional encoder with constraint length 7 and whose generation matrices (or shift register connections) are described in octal code. The upper generation matrix is 171_8 and the lower generation matrix is 133_8. This can be verified by studying the connections shown in Figure 8.

It is now understood that the output of the (2, 1, 7) convolutional encoder is a 2-bit value. It is required that this 2-bit value be serialized before passing it to the BPSK modulator (which implements 1-bit modulation). Hence, an *Unbuffer* block is used in Simulink this very purpose. The model is illustrated in Figure 9.

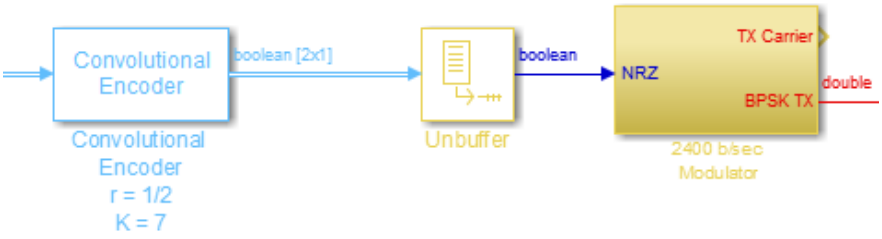

Figure 9. Simulink model of the (2,1,7) Convolutional Encoder and serializer (implemented with the Unbuffer block).

3.1.2. BPSK Modulator

In PSK, each bit is mapped to a distinct phase of a sinusoidal carrier. For BPSK, these phases are chosen to be 0 and 180 degrees with the transmitted signal $s(t)$, represented mathematically by Equation 2.

$$\boldsymbol{s(t)} = \begin{cases} \boldsymbol{A\sin(2\pi f_c t)}, & \boldsymbol{logic\,'1'} \\ \boldsymbol{A\sin(2\pi f_c t + \pi)}, & \boldsymbol{logic\,'0'} \end{cases}. \tag{2}$$

Then exploiting the fact that $\sin(2\pi f_c t + \pi) = -\sin(2\pi f_c t)$, the expression for the transmitted BPSK can be re-written as:

$$\boldsymbol{s(t)} = \begin{cases} \boldsymbol{A\sin(2\pi f_c t)}, & \boldsymbol{logic\,'1'} \\ \boldsymbol{-A\sin(2\pi f_c t)}, & \boldsymbol{logic\,'0'} \end{cases}, \tag{3}$$

where A is the Amplitude and f_c is the carrier frequency of the transmitted BPSK signal. From equation (7), the Simulink model of the BPSK modulator was designed to modulate a 2400 b/sec NRZ data stream by gating two antipodal sinusoidal carriers with amplitude, $A = 5\,V$ and a carrier frequency, $f_c = 4800\,Hz$. This implementation is illustrated in Figure 10.

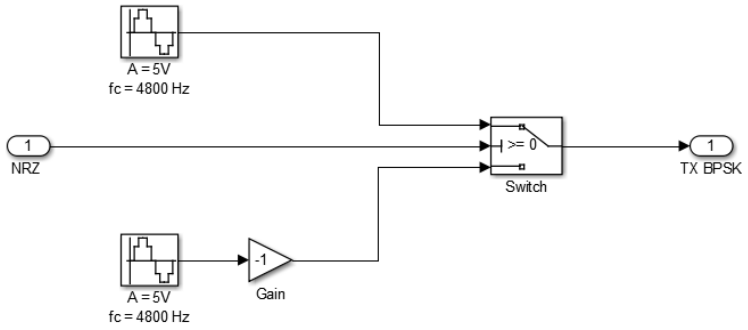

Figure 10. BPSK modulator that uses the Manchester data stream to gate two antipodal sinusoidal carriers that result in the BPSK modulated signal.

Figure 11 illustrates the operation of the BPSK modulator. When a binary '1' is input to the modulator, the positive sine wave is transmitted and when a binary '0' is input, the negative sine wave is transmitted.

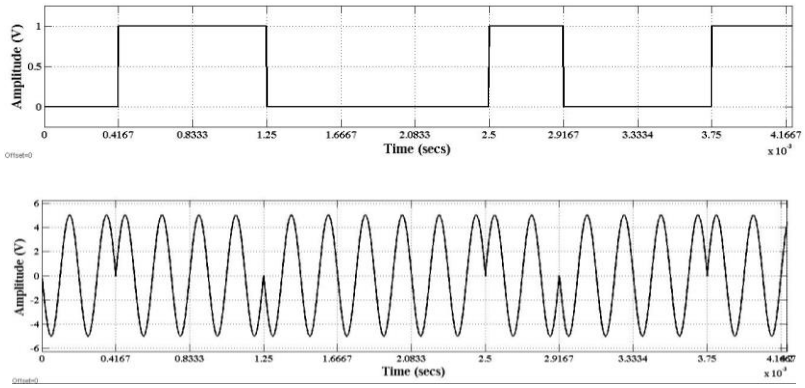

Figure 11. The transmitted BPSK signal has 180 degree phase shifts that correspond to logic level transition of the NRZ data stream.

3.1.3 AWGN Channel

The *AWGN Channel* in Simulink was used to model the propagation medium in the software simulation. The Simulink block operates in the *Signal to Noise Ratio (Eb/No)* mode. This mode requires the input signal power of the transmitted BPSK signal. Using the signal information provided in the previous section, we use the following equation to find this input signal power (Silage, 2009):

$$P = \frac{A^2}{T_b R_L} \int_{(i-1)T_b}^{iT_b} sin^2\big(k_p m_j(t) + \theta\big)dt = \frac{A^2}{2R_L} = \frac{5^2}{2} = 12.5\ W \qquad (i-1)T_b \le t \le iT_b \qquad j = 0, 1$$

where the amplitude A = 5 V, the bit period T_b = 1/2400 s, the load resistance R_L = 1 Ω, the phase shift k_p = π radians, and the phase offset $\theta = 0$ radians.

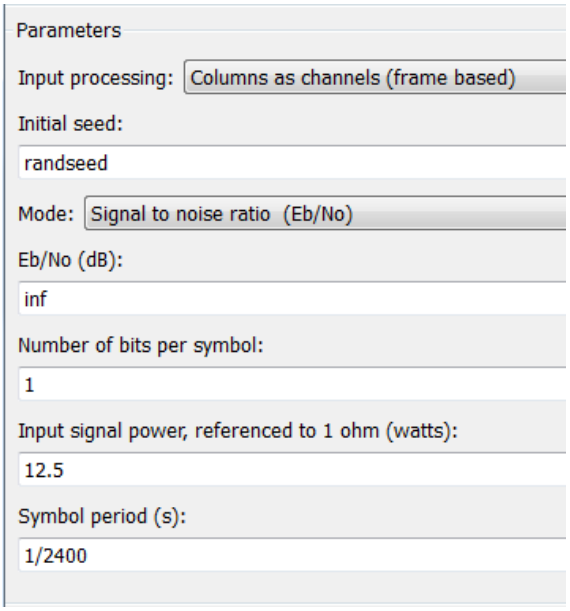

Figure 12. Configuration window for Simulink's AWGN block.

3.1.4 BPSK Demodulator: Carrier, Timing, Data Recovery & Soft-decision Encoding

Coherent demodulation of BPSK requires the receiver to be synchronized with the transmitter. As discussed in Section 1.3, synchronization requires two independent circuits. The first is the Costas loop carrier recovery circuit used for suppressed carrier reconstruction and data demodulation and the second is the Early-Late used for clock and data recovery. In this section we consider the theory, operation, and design of both circuits.

Costas Loop Design for Coherent BPSK Demodulation

The operation of the Costas Loop is explained using simple trigonometry and is illustrated in Figure 13. For simplicity, we assume the received signal is free of noise. This is only to illustrate loop operation.

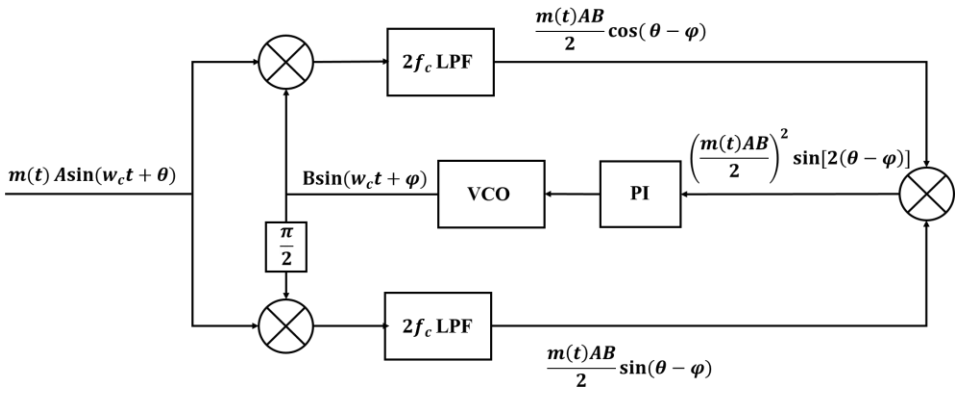

Figure 13. Time domain model of the Costas Loop carrier recovery and demodulation circuit. Its operation is described using trigonometric identities.

The received signal, $r(t)$ is the product of the data, $m(t)$ and the modulating carrier with phase, θ. This is equivalent to equation (3) where $(t) \in [1, -1]$.

$$r(t) = m(t) * A\sin(2\pi f_c t), \qquad m(t) \in [1, -1] \tag{4}$$

Referring to Figure 13, the received signal is split between the in-phase and quadrature arms where it is multiplied with the output of the local oscillator with amplitude, B and phase, φ. Thus the output of the multiplier in the in-phase arm is:

$$\boldsymbol{m(t)A\sin(w_ct+\theta)\cdot B\sin(w_ct+\varphi) = \frac{m(t)AB}{2}[\cos(\theta-\varphi)-\cos(2w_ct+\theta+\varphi)]}\,, \tag{5}$$

while the output of the multiplier in the quadrature branch is:

$$\boldsymbol{m(t)A\sin(w_ct+\theta)\cdot B\cos(w_ct+\varphi) = \frac{m(t)AB}{2}[\sin(2w_ct+\theta+\varphi)+\sin(\theta-\varphi)]} \tag{6}$$

Note that the VCO in the bottom arm is shifted by 90° to produce the quadrature component. Low pass filters (LPF) with cutoff frequencies at $2w_c$ remove the double frequency components leaving only the cosine and sine of the phase difference. The gains of the low pass filters are expressed as $|H(0)|$.

$$\boldsymbol{LPF\left\{\frac{m(t)AB}{2}[\cos(\theta-\varphi)-\cos(2w_ct+\theta+\varphi)]\right\}_{2w_c} = \frac{m(t)AB|H(0)|}{2}\cos(\theta-\varphi)} \tag{7}$$

$$\boldsymbol{LPF}\left\{\frac{\boldsymbol{m(t)AB}}{\boldsymbol{2}}[\boldsymbol{\sin(2w_c t+\theta+\varphi)+\sin(\theta-\varphi)}]\right\}_{\boldsymbol{2w_c}}=\frac{\boldsymbol{m(t)AB|H(0)|}}{\boldsymbol{2}}\ \boldsymbol{\sin\left(\theta-\varphi\right)} \tag{8}$$

The phase doubler on the right side of the Costas Loop multiplies the output of the in-phase and Quadrature LPF's. The resulting error signal, $e(t)$, drives the VCO towards phase lock.

$$\boldsymbol{e(t)=K_o m(t)\sin[2(\theta-\varphi)]}\ , \tag{9}$$

$$, \qquad \boldsymbol{K_o=\frac{(AB)^2|H(0)|^2}{4}}\ . \tag{10}$$

Assuming the VCO is locked in phase with the carrier of the transmitted BPSK signal ($\theta = \varphi$), then the output of the in-phase LPF from equation (7), reduces to $m(t)AB/2$ leaving only a scaled version of the demodulated data. In addition, when $\theta = \varphi$, the error signal, $e(t)$ reduces to zero since $\sin(0) = 0$. This implies the loop is locked in phase and frequency with $r(t)$.

However, fundamental questions arise like how long will it take the loop to lock? What is the lock range of the VCO? We can define specifications for the Costas Loop by applying concepts from classical control theory. Small angle approximations make it possible to linearize the Costas Loop in terms of phase. Figure 14 presents the Laplace domain model of the Costas Loop where it is assumed that $e(t) = K_o \sin[2(\theta - \varphi)] \approx K_d\, 2\psi$, where $\psi = \theta - \varphi$ is the phase error between the received signal and the VCO output.

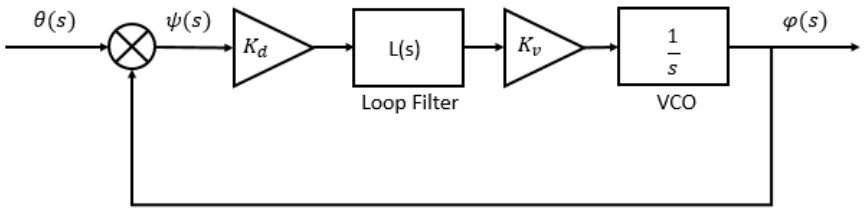

Figure 14. Linearized Costas Loop in the Laplace domain. Note that the $\boldsymbol{2f_c}$ filter is omitted in the linearized Laplace model because it is assumed that the double frequency component was filtered leaving only the phase error.

The open loop gain is expressed by K_o and includes the maximum amplitude of the received signal, the maximum amplitude of the VCO output and the gain of the arm filters. $L(s)$ Is the Loop filter that adjusts the phase error and determines the loops response characteristics and the VCO is modeled by an integrator that accumulates the phase with a sensitivity gain, K_v. Note that In Xilinx's Direct Digital Synthesizer (DDS) compiler, K_v is the phase increment value that dictates the output frequency, f_o (DDS datasheet citation). It is related to the output phase width, B_θ and the sampling frequency, f_s and expressed in equation (11).

$$K_v = \frac{2^{B_\theta} \cdot f_o}{f_s} \tag{11}$$

The closed loop transfer function of the Costas Loop in Figure 14 is then given by,

$$H(s) = \frac{\theta(s)}{\varphi(s)} = \frac{K_o K_v L(s)}{s + K_o K_v L(s)}, \tag{12}$$

And the steady error transfer function is accordingly,

$$\varphi(s) = \frac{s\theta(s)}{s + K_o K_v L(s)} \ . \tag{13}$$

In order for the Costas Loop to be able to track both phase and frequency steps, the appropriate loop filter L(s) must be chosen such that the steady error transfer has zero error for step ($\theta(s) = 1/s$) and ramp ($\theta(s) = 1/s^2$) inputs. The steady state errors are computed by application of the final value theorem to equation (13).

$$\textbf{Step Input}: \qquad \varphi(\infty) = \lim_{s \to 0} \frac{s^2\left(\frac{1}{s}\right)}{s + K_o K_v L(s)} = \mathbf{0} \tag{14}$$

$$\textbf{Ramp Input}: \qquad \varphi(\infty) = \lim_{s \to 0} \frac{s^2\left(\frac{1}{s^2}\right)}{s + K_o K_v L(s)} = \frac{\mathbf{1}}{K_o K_v L(s)} \tag{15}$$

The results of the final value theorem tell us that the loop filter must contain an integrator so that the steady state error for a ramp input approaches zero. Two common loop filters that accomplish this are the lead-lag filter and the proportional-integral filter (PI). In this design, the PI filter was chosen which takes the form,

$$L(s) = K_P + \frac{K_I}{s} \tag{16}$$

Where K_P is the proportional gain and K_I is the integral gain. After substitution of the PI filter back into equation (13), one finds that a steady error of $1/K_o$ still exists. However, by making the open loop gain sufficiently large, the steady state error can be minimized.

Next, the PI filter is substituted into equation (12) resulting in the following closed loop transfer function:

$$\boldsymbol{H(s) = \frac{K_d K_v K_P s + K_d K_v K_I}{s^2 + K_d K_v K_P s + K_d K_v K_I}} \quad . \tag{17}$$

Recognizing that equation (17) is a prototypical second order transfer function, the Costas Loop transfer function can be re-written in terms of the natural frequency w_n and the dampening ratio ξ (Crawford, 2008).

$$\boldsymbol{H(s) = \frac{w_n\left(1 + \frac{2\xi}{w_n}s\right)}{s^2 + 2\xi w_n + w_n^2}} \tag{18}$$

Where,

$$\boldsymbol{w_n = \sqrt{K_o K_v K_I}} \tag{19}$$

$$\boldsymbol{\xi = \frac{K_o K_v K_P}{2w_n}} \tag{20}$$

Lastly, the lock time (or settling time) and frequency lock range (in Hz) are expressed by equations (21) and (22) respectively.

$$\boldsymbol{T_{Lock} = \frac{4}{\xi \omega_n}} \tag{21}$$

$$\boldsymbol{f_{lock\ range} = f_{center} \pm \frac{K_o K_p K_v}{4\pi}} \tag{22}$$

In equation (22), $f_{lock\ range}$ dictates the upper and lower bounds of the frequencies that the VCO can successfully track and lock onto. Note that there will be a steady state error of $1/K_o$ as previously discussed.

Finally, using equations (18)-(20), (21) and (22) as a guide, a fixed point Simulink model of the Costas Loop was designed to meet the required lock time and lock range of $\leq 5\ ms$ and $\pm 200\ Hz$ respectively. A great deal of time was spent to achieve an accurate hardware representation of the Costas Loop because Simulink provides a GUI that makes optimization of loop performance much easier than Verilog.

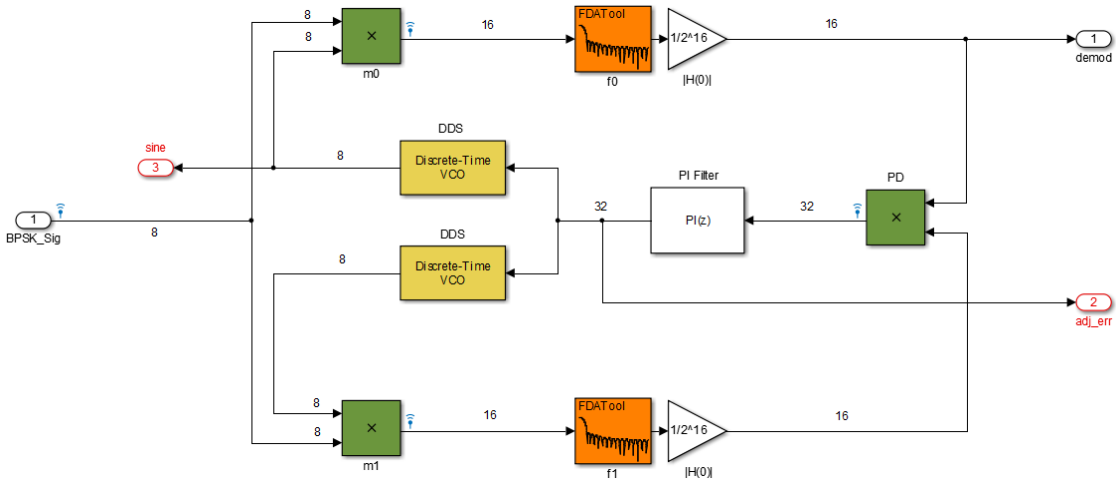

Figure 15. Fixed point Simulink model of the Costas Loop carrier recovery circuit. Yellow block represent instantiations of Xilinx Multiplier Blocks, green are Xilinx FIR Compiler blocks, and Orange is a DDS v4.0 compiler block.

The design of the Costas began by specifying the received BPSK signal width, VCO output signal width, VCO phase width, VCO center frequency, the Costas Loop sampling frequency, and the Costas Loop arm filter cutoff frequency. From these specification, the arm filters 'f0' and 'f1' are designed, followed by the computation of the filter gain, $|H(0)|$, VCO sensitivity gain, and then finally the proportional and integral gains, K_p and K_I that satisfy the loops requirements.

In this design, the received BPSK signal and the VCO output signal are specified as 8 bit signed integers with a maximum value of $2^{(8-1)} - 1 = 127_{10}$. This corresponds to the amplitudes A and B in equation (10). The VCO phase width determines the resolution of the output sinusoid in addition to the VCO sensitivity gain. This was chosen to be 32 bits in order to maximize resolution.

Received Signal Max. Amplitude, A	8 bits: 127_{10}
VCO Output Max. Amplitude, B	8 bits: 127_{10}
VCO Phase Width, B_θ	32 bits
VCO center Frequency, f_c	4800 Hz
Costas Loop Sampling Frequency, f_s	2 MHz
Arm Filter Cut-Off Frequency, $2f_c$	9600 Hz

Table 3. Initial Specifications for the design of the Costas Loop carrier recovery circuit.

The VCO center frequency is 4800 Hz so that when the Costas Loop is locked in phase and frequency

with the received BPSK signal, the data is coherently demodulated. The sampling frequency of the Costas Loop is 2 MHz and from equations (7) and (8), the cutoff frequencies are specified to be 9600 Hz. Table 3 summarizes the initial Costas Loop design specifications.

The in-phase (I) and quadrature (Q) arm filters in the Costas Loop were designed using Simulink's FDA tool. The design uses a 31 tap, direct-form FIR filter with a cutoff frequency of 9600 Hz. The coefficients were double precision values that were scaled to 12 bit unsigned integers. This is required because Xilinx's FIR Compiler only recognizes integer coefficients. Figure 16 displays the magnitude and phase response of the I and Q arm filters.

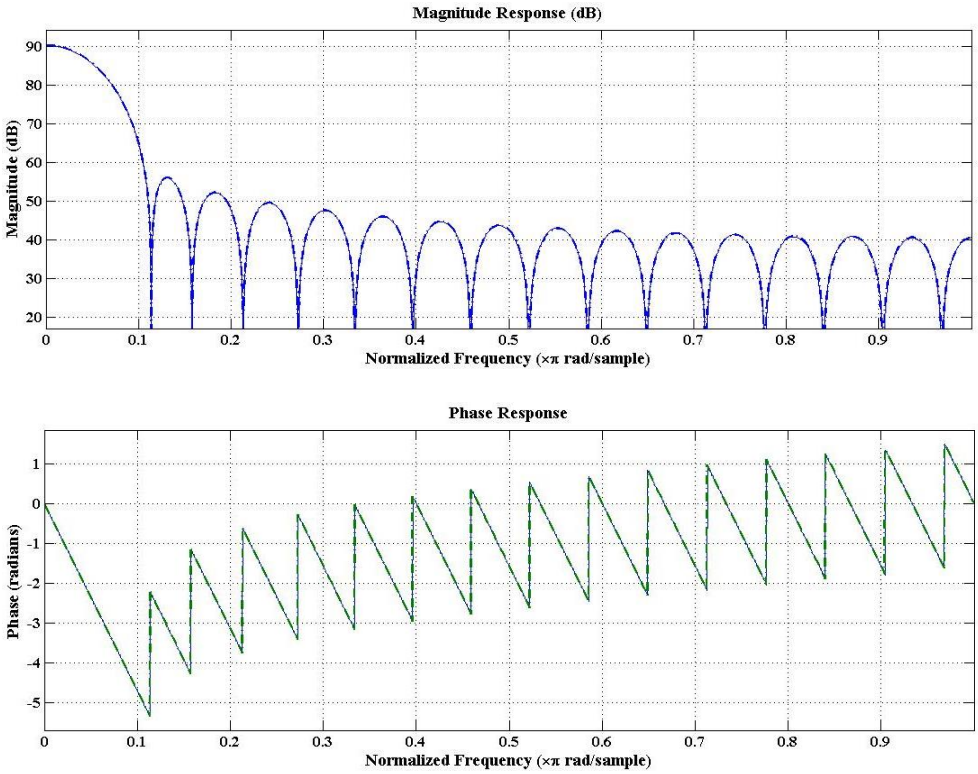

Figure 16. Costas Loop's I and Q arm filter's magnitude and phase response.

Next, the required output register size is computed. In Figure 15, 'm0' and 'm1' are 8x8 bit signed multipliers with a signed 16 bit product. It follows then that the filter inputs are also signed 16 bit registers with maximum values of $2^{(16-1)} - 1$. The required output register size is then computed as the summation of integer coefficients, multiplied by the maximum input value ($2^{(16-1)} - 1$). Using the integer coefficients from the above design requires an output register size of 32 bits. However, referring back to the Costas Loop in Figure 15, this requires a 31x31 bit multiplier for 'PD' while the DSP48 in the Spartan 6 only supports 18x18 bit multipliers (DSP48 Datasheet). The problem is mitigated by truncating the 16 LSBs of the filter output. In the Simulink model in Figure 12, this truncation is expressed as a gain of $1/2^{16}$ which is equivalent to an arithmetic right shift by 16 bits. This is also denoted as, $|H(0)| = 1/2^{16}$ in our mathematical model. After $|H(0)|$ was specified, the open loop gain, K_o was computed from equation (10) and found to be $K_o = 0.0161$.

The VCO sensitivity gain was then computed from equation (11) and the specifications in Table 3. The result is $K_v = 1.0737 \times 10^7$. Having now computed the open loop gain and the VCO gain, the

proportional and integral gains, K_p and K_I are found by the simultaneous solution of equations (19) – (22). The resulting gains are summarized in Table 4. Just like the filter coefficients, the proportional and integral gains must be integers to adhere to our fixed point design. Although there are several techniques for scaling, this design approximated proportional and integral gains as an arithmetic shifts.

Arm Filter Gain, $\lvert H(f)\rvert$	$1/2^{16}$
Open Loop Gain, K_o	0.0161
VCO Sensitivity Gain, K_v	1.0737×10^{7}
Proportional Gain, K_P	$0.0160 \approx 1/2^{6}$
Integral, K_I	$0.0041 \approx 1/2^{8}$

Table 4: Gains required to meet the desired Lock time and Lock Range

Thus referring to Table 4: Gains required to meet the desired Lock time and Lock Range, the proportional gain 0.0160 which is approximately $1/2^6$. Likewise, the integral gain resulted in 0.0041 which is approximately $1/2^8$. The benefit of this approach is that the implementation of PI filter in Verilog can be done using only adders and shift registers instead of multipliers. Figure 17 and Figure 18 display the results of two particular simulations. . Figure 17demonstrates successful carrier tracking when the receiver is 60 degrees out of phase with the transmitter and Figure 18 demonstrates tracking when there is a 200 Hz frequency difference between the receiver and transmitter.

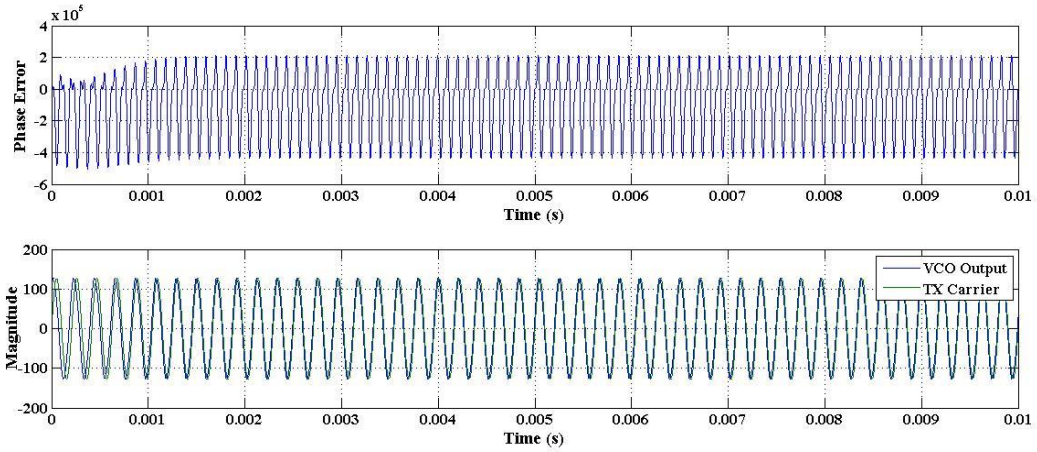

Figure 17. (Top) Costas Loop phase error and (Bottom) VCO output overlaid on the 4800 Hz TX carrier. Phase offset is 60 degrees and VCO center frequency is 4.8 kHz.

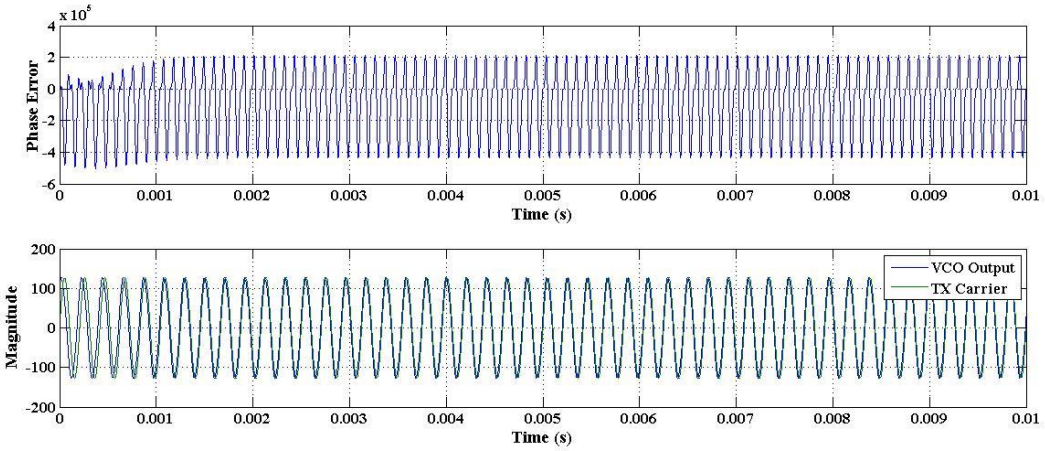

Figure 18 (Top) Costas Loop phase error and (Bottom) VCO output overlaid on the 4800 Hz TX carrier. Phase offset is $\boldsymbol{\pi/3}$ and VCO center frequency is 5 kHz.

The results in Figure 17 and Figure 18 illustrate that the Costas loop meets the proposed design requirements. The loops lock time for both phase and frequency steps are less than 3 ms and is capable of tracking 200 Hz frequency steps. In addition, as predicted previously, Figure 18 shows a small but negligible steady error as predicted earlier. This is corroborated in Figure 19 which displays the demodulated signals for each simulation.

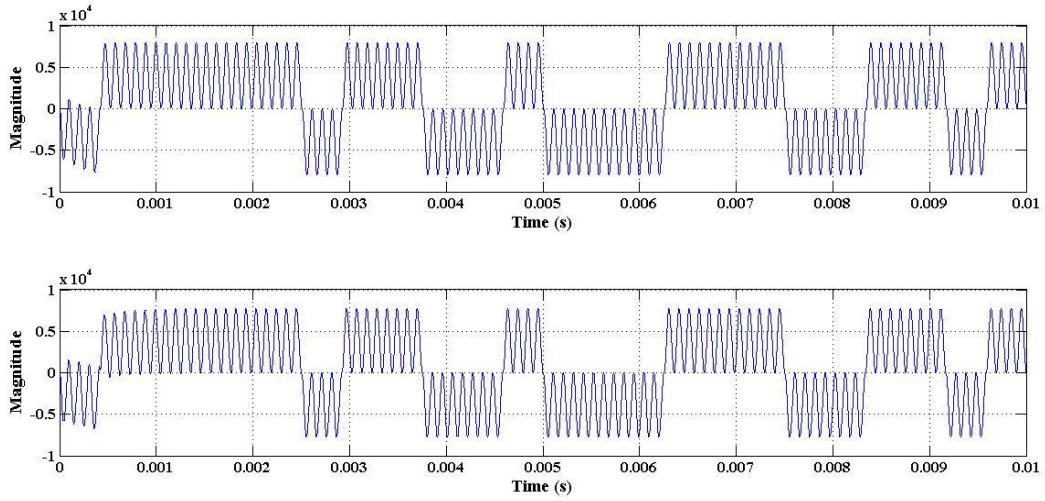

Figure 19. (Top) Demodulated data for 60 degree phase step and (Bottom) demodulated data for a 200 Hz frequency step.

Timing Recovery Using Early-Late Gate Synchronizer

The synchronization between the transmitter and the receiver is done using the Early-Late Gate Timing Recovery circuit, among the several other circuits available, the Early Late Gate has proven to be the

ideal design in communication links. The Early-Late Gate provides the opportunity to constantly ameliorate synchronization through its feedback components and its loop filter. As section 1.3.4 explains, the error between the early clock and the late clock is to converge to zero in ideal conditions. For the error to converge to zero at a certain time t_s, the Early Late Gate depends on its loop filter component in circuitry. This component is used to minimize the settling time of the circuit to synchronize the received signal. Therefore, based on those facts, it was found beneficial to implement a PID controller for achieving an appropriate settling time. Figure 7, would then consist of a PID controller for its loop filter, while the remaining components will remain the same.

After obtaining a zero error rate from the Early Late Gate, the early and late clocks must operate within a specified region which allows the integrators to accumulate an equal amount of energy from the two branches. This region is commonly referred to as the *correct receiver timing* while the *early/ late receiver timing* are known to detriment the error between the early and late gates. Figure 20 illustrates the desired condition for the three samples to operate in, note that the energy accumulated by the integrators will be equal. Before it locks itself into the *correct timing* the circuit first adjusts its clock using the PID controller. The controller, makes use of the Proportional and Integral gains to drive the output clock from the error signals. The PI controller gains were determined heuristically to be P = 0.15 and I = 0.25.

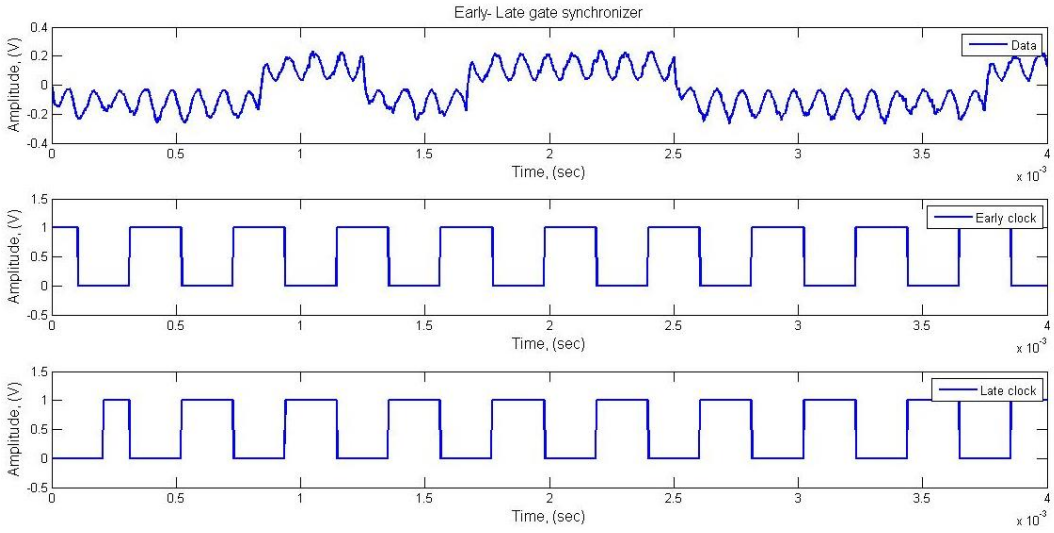

Figure 20. Ideal timing between the Early and Late branch

Figure 21 depicts the Early-Late Gate's implementation in Matlab Simulink, since the signal is sampled at a rate of 480 kHz, the demodulated signal will be composed of 200 samples per period hence (480 kHz/ 2400 Hz = 200 samples/period), a delay of 100 samples on the clock (at the late gate) is equivalent to a half-bit delay and is then used to control the *integrator* and the *sample and hold blocks* of the late branch. And finally, the Voltage Controlled Clock (VCC) is implemented with a traditional VCO that is later converted into a clock signal using a bang-bang controller and a sample and hold. The clock signal from the VCO drives the integrators and sample-and-holds in the early and late branches which closes the loop in the Early-Late gate circuit.

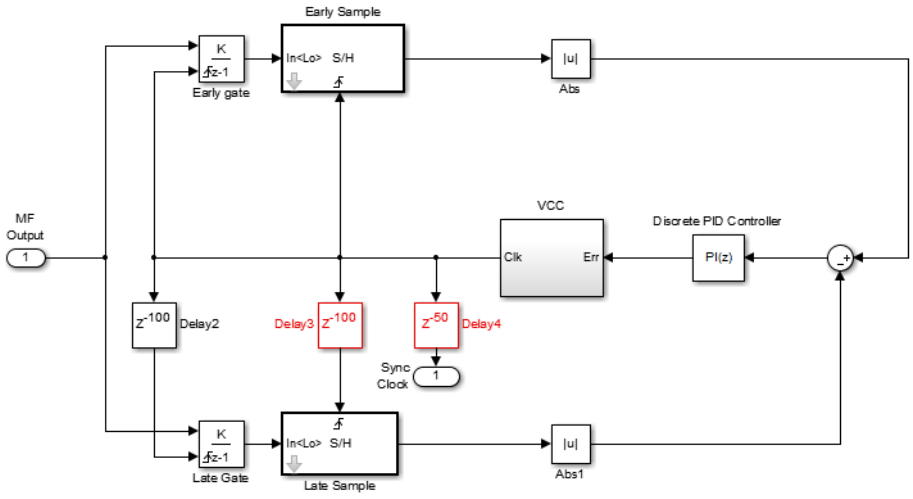

Figure 21 Simulink implementation of the Early- Late gate timing synchronizer

Within the *correct timing region*, the error between the late and early branch is expected to be zero as Figure 22 illustrates, both branches will integrate different regions of the waveform, but eventually, the amount of energy will sum to the same quantity. The error is then obtained by the subtraction of the two signals, the error is kept small by the use of the PID controller which drives the clock to lock onto the edges of the signal. Figure 22 shows the error rate of the Early- Late Gate circuit, by first observation, it can be seen that the energy at the two signals are the same except for very small instances where one branch peaks more than the other. By tuning the PID controller, those instances are weighted accordingly to drive to the VCO which provides the synchronized clock.

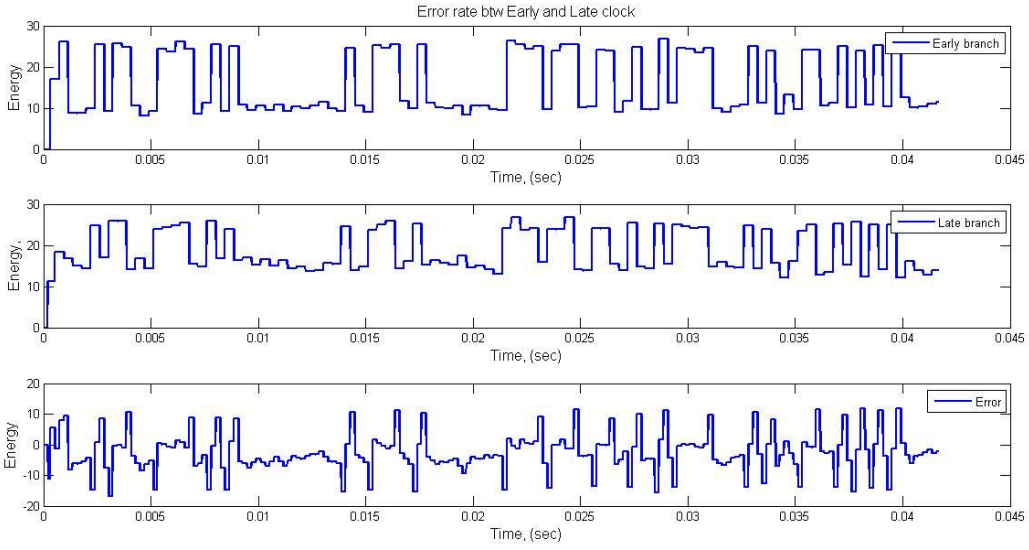

Figure 22. (Top) Energy accumulated in early branch. (Middle) Energy accumulated in late branch. (Bottom) Error rate between the Late and Early branch

Soft-decision Encoding

NOTE: This circuit was not used in the final hardware implementation of this senior design project because of implementation issues with the Viterbi Decoder. Details are explained in the evaluation section.

Instead of implementing hard-decision decoding and producing a logic 0 or 1 as an output, the output of the matched filter is a 3-bit scalar quantization (SQ) encoder. The range of the 3-bit output of the SQ encoder represents the confidence level of the demodulated signal actually being a logic 0 or 1 (shown in Table 5). This indecisiveness justifies the term "soft-decision" decoding. Figure 23 shows the Simulink model of the soft-decision encoder used to implement the 3-bit scalar quantization encoding.

Input Value	Confidence
0	Most confident zero
1	Second most confident zero
2	Second least confident zero
3	Least confident zero
4	Least confident one
5	Second least confident one
6	Second most confident one
7	Most confident one

Table 5. This table shows the output of the *Scalar Quantization Encoder* which is used by the *Viterbi Decoder*.

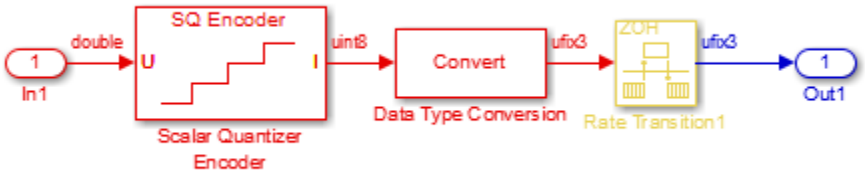

Figure 23. Scalar Quantization Encoder scalar quantizes BPSK demodulator output into a 3-bit value. This 3-bit "soft-decision" output is used by the *Viterbi Decoder*.

3.1.5 Forward Error Correction: Soft-decision Viterbi Decoding

There is a 6-bit buffer at the output of the soft-decision BPSK demodulator. The purpose of the buffer is

to collect two of the 3-bit data and input it into the Viterbi decoder. Recalling that the code rate of this convolutional code is 1/2, the buffer essentially undoes what the 2-bit serializer did at the transmitter.

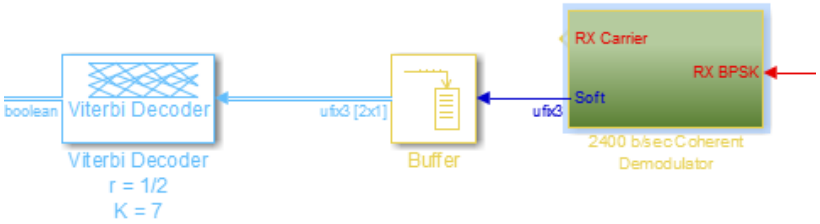

Figure 24. Simulink model of the Buffer that takes parallel data and converts it to serialized data for the Viterbi Decoder

The soft-decision Viterbi decoder uses Hamming distance and maximum likelihood to decode the convolutional encoding. The ½ code rate convolutional code with a constraint length of 7 has a *free distance* d_f of 10 (Sklar, 2001). Essentially, free distance is the minimum Hamming distance between the all-zero path and any arbitrary path that diverges and remerges with the all-zero path (Sklar, 2001). It is essentially a measure of how well a convolutional code corrects errors. The higher the free distance, the better the convolutional code is at correcting random errors. A Viterbi decoder can correct up to *t* random errors (Sklar, 2001):

$$\boldsymbol{t = \left\lfloor \frac{d_f - 1}{2} \right\rfloor} \tag{23}$$

Hence, the ½ code rate convolutional code with constraint length 7 can correct up to 4 random errors in a received signal. The upper-bounded probability of bit error for the (2, 1, 7) convolutional code with BPSK and soft-decision decoding (Sklar, 2001) is expressed as follows:

$$\boldsymbol{P_b \leq \frac{Q\left(\sqrt{\frac{7E_b}{N_0}}\right)}{\left[1 - 2e^{\left(\frac{-E_b}{2N_0}\right)}\right]^2}} \tag{24}$$

This upper-bounded probability of bit error is related to an upper-bounded coding gain (dB) expressed as follows (Sklar, 2001):

$$coding\ gain\ \leq 10\ \log_{10}(rd_f) \cong 7\ dB$$

where the code rate *r* is:

$$r = \frac{k}{n} = \frac{1}{2} \tag{25}$$

The result of the coding gain inequality means that BPSK with soft-decision Viterbi decoding ('2,1,7' code, ½ code rate, constraint length 7) offers at most 7 dB of coding gain compared to un-coded BPSK.

The *Viterbi Decoder* uses a *traceback length* of 48, which is within the recommended range for a convolutional code with constraint length K = 7 (Sklar, 2001). The traceback length is a metric for path memory since the convolutional code depends on both currently inputted data and recently inputted data.

3.1.6 Simulation Results

One of the main goals of this senior design project is to implement BPSK modulation with soft-decision Viterbi decoding via both computer simulation (Simulink) and hardware implementation (FPGA). From the computer simulation models, we are able to determine if the observed BER performance results correlate with theory. If the results do compare with theory, the next goal would be to implement the models in FPGA hardware.

The following graph (Figure 25) shows the BER performances for several targeted amateur satellite telemetry communication schemes. The graph shows that there is approximately a 3 dB improvement to the perceived SNR when using BPSK instead of BFSK. The results of simulation for BPSK with soft-decision Viterbi decoding provides a maximum of 7 dB coding gain when compared to un-coded BPSK. This result matches with the theoretical coding gain inequality provided in the previous section. Additionally, there is approximately 2 dB of coding gain when using soft-decision Viterbi decoding instead of hard-decision Viterbi decoding (Viswanathan, 2013). The curve for BPSK with hard-decision Viterbi decoding satisfies this fact by consistently performing 2 dB worse than soft-decision Viterbi decoding. These BER performance results serve as the basis for determining how much power consumption is reduced by opting to use BPSK with soft-decision Viterbi decoding instead of just BFSK (Bell 202) or BPSK in amateur radio satellite telemetry communications.

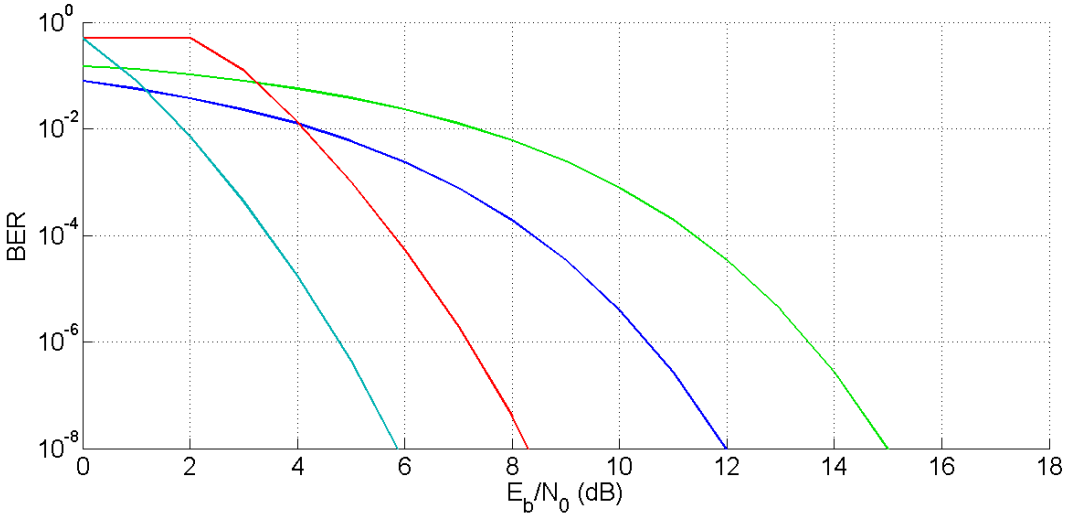

Figure 25. BER performance for several amateur satellite telemetry communications schemes over AWGN. The green curve is the BER performance for BFSK. The dark blue curve is the BER performance for BPSK. The red curve is the upper-bounded BER performance for hard-decision BPSK with the (2, 1, 7) convolutional code. The teal curve is the upper-bounded BER performance for soft-decision BPSK with the (2, 1, 7)

convolutional code.

3.2. Hardware Implementation using ISE Project Navigator

This section discusses the hardware implementation of the software model showcased in the previous section. The goal is to conserve all of the results of the software model during hardware implementation. With a sufficiently developed software model, it should be relatively effortless to realize the software model in hardware. Hence, this section will showcase the hardware architecture of this senior design project and the alterations that were needed to realize the software model in hardware.

The hardware implementation of this senior design project includes using a desktop PC as a bit error tester (BERT) and an FPGA as the design under test (DUT). The role of the BERT is to transmit 10,000 bits of data to the DUT, wait for the DUT to process the data, and then receive 10,000 bits to begin the BER calculation. The RS-232 serial communications scheme is used to interface the BERT to the DUT. The RS-232 parameters include 1200 b/sec data rate, 8 data bits, no parity bit, one stop bit, and no hardware or software flow control. The role of the DUT is to implement the two 10,000 bit storage buffers and the digital communication loopback. One storage buffer collects 10,000 bits from the BERT. The other storage buffer collects the 10,000 bits from the digital communication loopback before being returned to the BERT. The digital communication loopback consists of the BPSK modem with forward error correction and the propagation channel model (AWGN). The next figure (Figure 26) depicts the high-level system architecture of this senior design project.

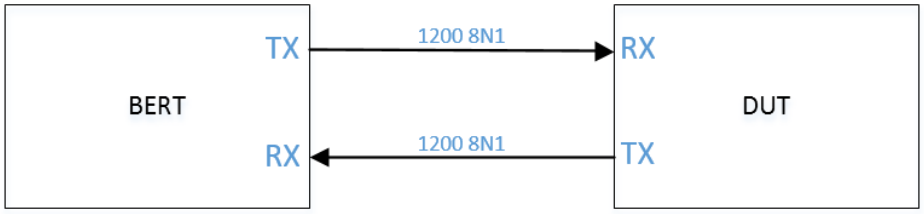

Figure 26. High-level system architecture of this senior design project. A desktop PC is used to implement bit error test (BERT) functionality while an FPGA is used to implement the design under test (DUT). The RS-232 serial communication scheme is used to interface the BERT to the DUT. The chosen RS-232 parameters are 1200 b/sec bit rate, 8 bits of data bits, no parity bit, one stop bit, and no software or hardware flow control.

The primary components of the BERT will be discussed in the next section, *3.2.1 Serial Terminal Program & Software BER Calculation Script*. The DUT is realized using two products from an electronics company named Trenz Electronics. The two products are the TE0304 Carrier Board and the TE0630 FPGA Module. An example of this setup is shown Figure 27. An FPGA module is sitting atop of the carrier board. The carrier board is equipped with an abundance of I/O. Important to this senior design is the DCE RS-232 port located at the bottom of Figure 27.

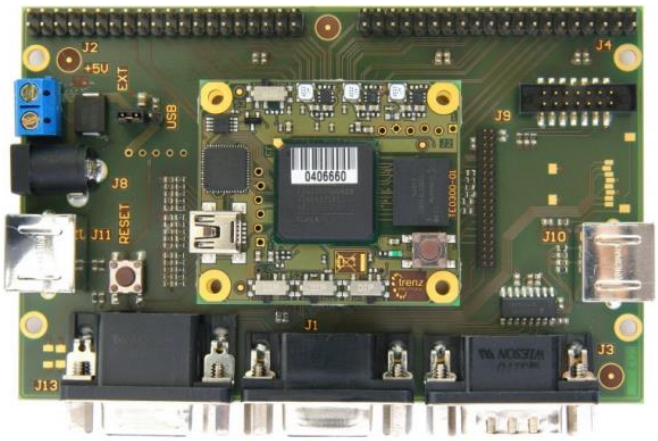

Figure 27. Trenz Electronics products used for hardware implementation of the DUT of this senior design project. The TE0304 carrier board is shown supporting an FPGA module (not the TE0630). The DCE RS-232 port is located at the bottom of the image. The port is used to interface to the BERT.

Figure 28 shows the actual DUT set up for use. The RS-232 cable is clearly interfaced at the top of the image.

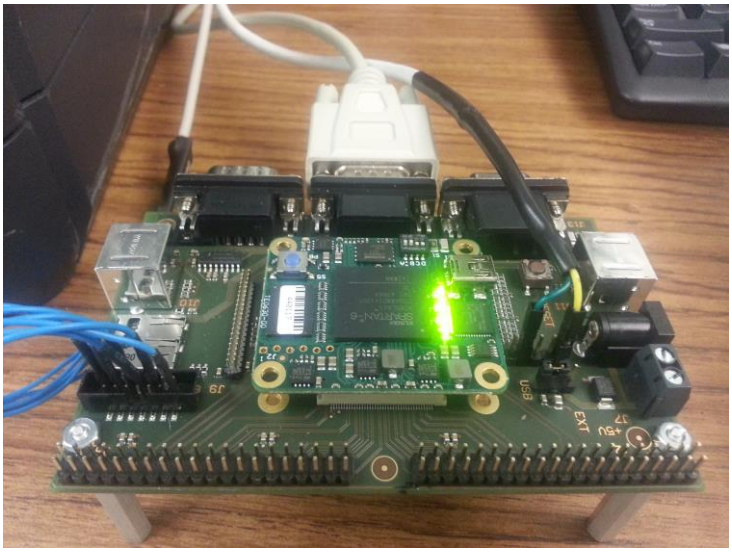

Figure 28. Actual setup of DUT for this senior design project. It is shown interfaced to the BERT (PC; left) via RS-232 serial communication.

The DCE port on the TE0304 carrier board connects to the FPGA module via two board-to-board (B2B) connectors. Figure 29 shows the high-level system architecture of the DUT as implemented on the FPGA module. The left side of the figure represents inputs, while the right side represents outputs.

Figure 30 depicts the top-level I/O wrapper module (top I/O module) of the DUT. The FPGA pins shown in Figure 30 are clearly shown as the inputs and outputs to the top I/O module. The top I/O module consists of several sub-modules. Two of the sub-modules (rcvr.v, txmit.v) interface with the BERT via

the RS-232 serial communication scheme. The sub-module satcom.v encapsulates the entire digital communication loopback consisting of the two 10,000 bit storage buffers, the BPSK modem with forward error correction, and the propagation channel model (AWGN). There are two sub-modules dedicated to DAC interfacing (for testing purposes).

The remainder of this section will delve deeper into the hardware implementation of the BERT and DUT. It begins with a discussion of the primary components of the BERT in section, *3.2.1 Serial Terminal Program & Software BER Calculation Script*.

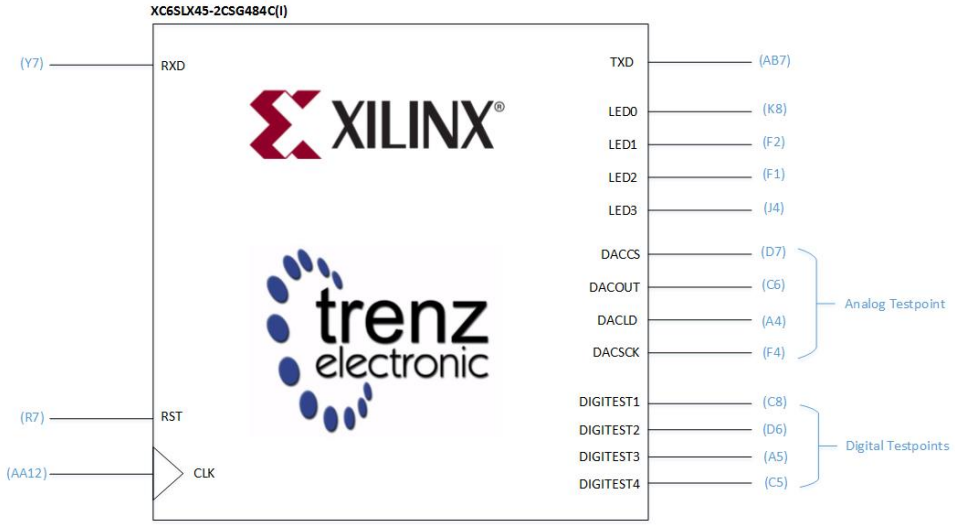

Figure 29. High-level system architecture of the DUT of this senior design project. Shown is a functional block diagram of the Xilinx Spartan 6 XC6SLX45-2CSG484C(I) located on the TE0630 FPGA module. The FPGA pins used are highlighted and associated with the internal functionalities. Namely, there are pins used to interface to the DCE RS-232 port of the TE0304 carrier board, 100 MHz system clock, system reset, user LEDs, and PMod™ ports. The PMod ports are used to interface with a DAC or logic analyzer for testing purposes.

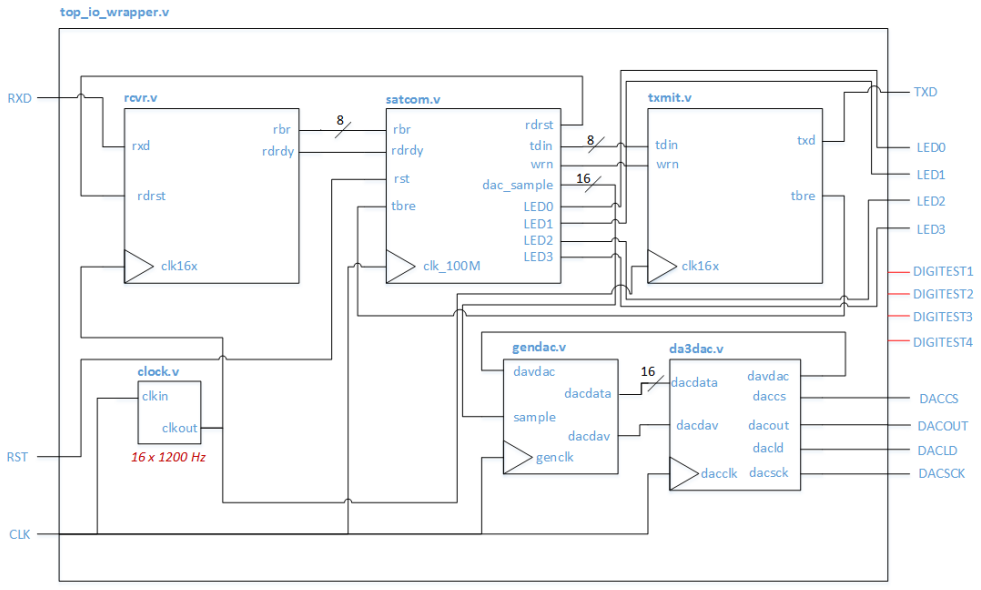

Figure 30. The top-level I/O wrapper module for the DUT. This module interfaces to the various I/O available to the FPGA including RS-232 ports, 100 MHz system clock, system reset, four user LEDs, and two PMod™ ports for interfacing to a DAC or logic analyzer. The sub-modules implement RS-232 interfacing (rcvr.v, txmit.v), digital communication loopback (satcom.v), and DAC interfacing (gendac.v, da3dac.v). The four digital testpoint signals (DIGITEST1, etc.) are reserved for testing/investigating any digital signal throughout the entire DUT. Access to ChipScope Pro software logic analyzer or Hardware-In-The-Loop software would have eliminated the need for the DAC and PMod™ interfacing.

3.2.1 Serial Terminal Program & Software BER Calculation Script

A serial terminal program was used to interface the BERT (PC) to the DUT (FPGA) via the RS-232 serial communication scheme. The RS-232 parameters used were 1200 b/sec data rate, 8 data bits, no parity bit, one stop bit, and no software or hardware flow control. The purpose of the BERT is to provide the DUT 10,000 bits for processing and then subsequently store the 10,000 bits returned from the DUT. The serial terminal program is used to perform this feat. Specifically, the TeraTerm Web 3.1 serial terminal program was used in this senior design project. The software was setup to implement the RS-232 protocol mentioned above.

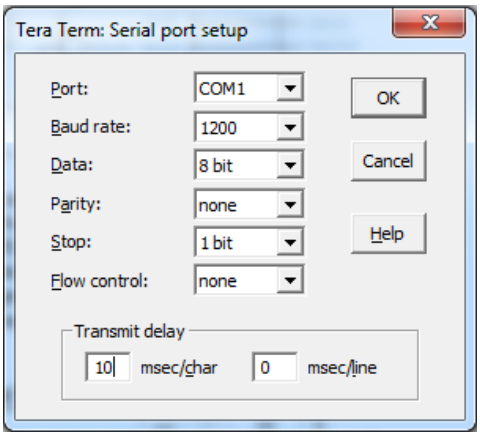

Figure 31. RS-232 serial terminal setup parameters for 1200 8N1 communication between BERT and DUT. A 10 millisecond delay is used between each ASCII character transmission since no flow control is used.

Figure 32 illustrates the transmit delay between each ASCII character transmitted from the BERT to the DUT. The size of the transmit delay is a function of how long it takes the DUT to receive (then store) and transmit (then re-transmit) an ASCII character without flow control.

Figure 32. Two ASCII characters being transmitted from the BERT to the DUT using 1200 8N1 RS-232 communication. A 25 ms transmit delay occurs between each ASCII character transmission, but the delay can be reduced. How much of a reduction depends on how quickly the DUT can receive or transmit a single ASCII character.

The TeraTerm software also has the capability to send text files. This feature was used to transmit a 10,000 bit file to the DUT. An ASCII character is 8 bits in size. 1250 ASCII characters are 10,000 bits in size. Hence, 1250 ASCII characters comprise the sent file. Further, this means that it takes approximately 23 seconds to transmit the 10 kb file to the DUT:

$$(1250 \times 10 \text{ ms}) + 1250 \times \left(\frac{1+8+1}{1200} \text{ s}\right) \cong 23 \; seconds$$

It takes 23 seconds also to receive the 10 kb of data from the DUT. Hence, it takes approximately 46 seconds to fully transmit and receive the 10 kb file. Keep in mind, this 46 seconds does NOT include the time required to propagate the 10 kb file through the digital communication loopback of the DUT. This processing time will be further investigated in the upcoming sections.

3.2.2 10K-bit Receive and Transit Storage Buffers

As shown in Figure 32, RS-232 serial communication is an asynchronous communication method. Even though our scheme may use 1200 b/sec bit rate, the actual data rate is not 1200 b/sec. Depending on the size of the transmit delay between ASCII character transmission, the data rate can be much lower than the RS-232 bit rate. Hence, it is necessary for the DUT to store the received data bits in a buffer until all 10,000 bits are received by the BERT. Once the 10,000 bits are stored, then a continuous data stream (1200 b/sec) can flow into the digital communication loopback. This is the description of the 10K-bit receive storage buffer. A transmit buffer has similar functionality. Its role is to collect the 10,000 bits that entered the digital communication loopback. Once the 10,000 bits are stored, the bits are then asynchronously transmitted back to the BERT for the BER calculation.

Figure 33 shows the RS-232 receiver module (rcvr.v) of the DUT which interfaces to the RS-232 transmitter of the BERT. The receiver module (rcvr.v) processes the incoming RS-232 signal from the BERT, resulting in an 8-bit ASCII character. This 8-bit character is then provided to satcom.v for storage in a 10,000 bit buffer. The role of this 10K-bit buffer is to *asynchronously* store 10,000 bits of data (1250

ASCII characters) from the BERT before releasing the stored bits into the digital communication loopback at a continuous data rate of 1200 b/sec. Once the 10K-bit buffer is full, it will not store any new data until all 10,000 bits are transmitted into the digital communication loopback.

Further, the 10K-bit transmit buffer must know when to begin storing bits from the digital communication loopback. That is, when the 10K-bit receive buffer begins to pass bits into the digital communication loopback, the 10K-bit transmit buffer must know how long of a period to wait before the bits propagate to its input port. Consequently, a transmit delay feature is implemented by the 10K-bit transmit. This feature works as follows:

1. The 10K-bit receive buffer fills up with 10,000 bits.
2. The 10K-bit receive buffer alerts the 10K-bit transmit buffer to this event.
3. The 10K-bit receive buffer begins passing the 10,000 stored bits into the digital communication loopback.
4. The 10K-bit transmit buffer waits a fixed amount of time upon receiving the alert.
5. The 10K-bit transmit buffer fills up with 10,000 bits.

A controller module (buffercontrol.v) is used to implement the handshake signaling interface between the RS-232 receive and transmit modules (rcvr.v, txmit.v) and the 10K-bit receive and transmit storage buffer modules (rcvbuffer.v, txmitbuffer.v). The next figure depicts how this all fits together.

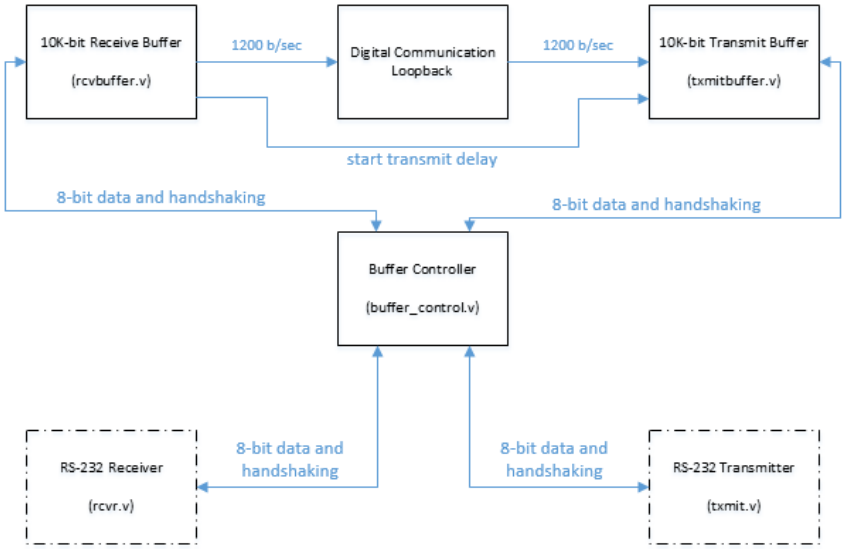

Figure 33. A high-level block diagram depicting the interfacing between the RS-232 receiver and transmitter modules (rcvr.v, txmit.v) and the 10K-bit receive and transmit storage buffers (rcvbuffer.v, txmitbuffer.v). The 8-bit character received by the RS-232 receiver module (rcvr.v) is passed to the buffer controller module (buffer_control.v). The character is then passed from the buffer controller module to the 10K-bit receive storage buffer. Once 1250 characters (10,000 bits) are stored in the 10K-bit receive buffer, the 10K-bitt transmit buffer is alerted and starts a transmit delay timer. Once the transmit delay time passes and the 10K-bit transmit storage buffer stores 10,000 bits, it passes the bits as 8-bit characters to the buffer controller module. From there, the buffer controller passes the 8-bit character to the RS-232 transmitter module (txmit.v).

Figure 34 is a human-readable flow chart of the state machine within the buffer controller module (buffer_control.v). The state machine consists of eleven states. Six states (colored blue) control the interface between the RS-232 receiver module (rcvr.v) and the 10K-bit receive storage buffer module (rcvbuffer.v), one state (colored green) indicates that 10,000 bits are passing through the digital communication loopback, and four states (colored red) control the interface between the 10K-bit transmit storage buffer module (txmitbuffer.v) and the RS-232 transmitter module (txmit.v). Three of the user LEDs on the FPGA module are used to identify which of the three group of states the DUT is currently in. LED0 is associated with the six states, LED1 is associated with the one state, and LED2 is associated with the four states. A fourth user LED (LED3) is used to indicate whenever the RS-232 transmit or receive modules (txmit.v, rcvr.v) are actively processing data.

State 0: of the state machine (Figure 34) is entered upon the start of running FPGA image or upon system reset. This state is used to reset the RS-232 receiver module and enable it to detect a new RS-232 packet. This state is entered whenever a new RS-232 packet is needed by the 10K-bit receive storage buffer module. This state activates LED0 to indicate that the RS-232 receiver module is working with the 10K-bit receive storage buffer to store 10,000 bits of data from the BERT. **State 1** is used to un-reset the RS-232 receiver module. **State 2** is used to wait for a new RS-232 packet. If a new packet is asynchronously detected, state 3 is entered, otherwise the state machine remains in state 2. **State 3** passes the received packet to the 10K-bit receive storage buffer module. State 3 is then used to determine when the 10K-bit receive storage buffer module is done storing the received packet. If the packet is not yet stored, the state machine stays in state 3. If the packet is stored and the 10K-bit buffer is not full, the state machine enters state 4. If the 10K-bit buffer is full, the state machine enters state 5. **State 4** is used to implement handshake signaling between the RS-232 receiver module and the 10K-bit receive storage buffer module. When the handshake is completed, the state machine enters state 0. **State 5** is used to dispense the 10K-bit receive buffer when it becomes full. When the 10K-bit receive buffer is emptied, an alert signal is sent from the 10K-bit receive buffer module to the 10K-bit transmit buffer module. This signal is a part of the transmit delay functionality of the 10K-bit transmit buffer module. When the 10K-bit receive buffer is emptied, the state machine enters state 6.

State 6: is used to indicate that the 10,000 received bits are being passed through the digital communication loopback. This state deactivates LED0 and activates LED1 to indicate that the 10K-bit transmit buffer module has not yet received 10,000 bits. Once the 10K-bit transmit buffer module receives 10,000 bits, the state machine enters state 7.

State 7: is used to indicate whether the RS-232 transmitter module is busy transmitting an ASCII character or not. If the former is true, the state machine enters state 8, otherwise the state machine remains in state 7. **State 8** is used to pass a stored character (8 bits) from the 10K-bit transmit buffer to the RS-232 transmitter module. If the 10K-bit transmit buffer is empty, the state machine enters state 0. If 8 bits of data is still being prepared by the 10K-bit transmit buffer, the state machine remains in state 8. If 8 bits of data is available for the RS-232 transmitter module, the state machine enters state 9. **State 9** is used to implement the handshake signaling between the buffer controller module and the 10K-bit transmit buffer module. Once a successful handshake occurs, the state machine enters state 10. **State 10** is used to implement the handshake signaling between

the buffer controller module and the RS-232 transmitter module. Once a successful handshake occurs, the state machine enters state 7.

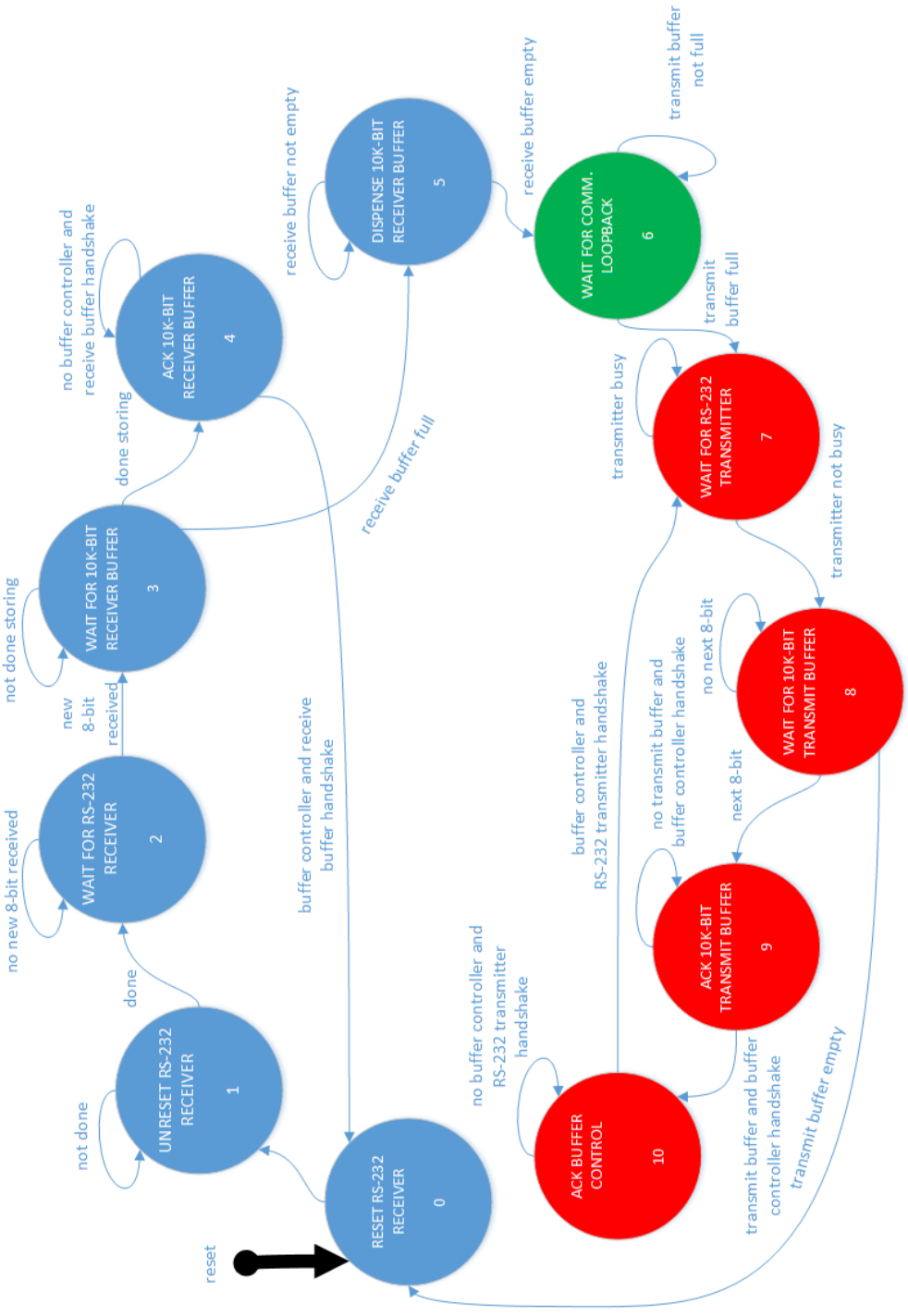

Figure 34. Human-readable flowchart of the state machine in the buffer controller module (buffer_control.v). The state machine

consists of eleven states. Six states (colored blue) control the interface between the RS-232 receiver module (rcvr.v) and the 10K-bit receive storage buffer module (rcvbuffer.v), one state (colored green) indicates that 10,000 bits are passing through the digital communication loopback, and four states (colored red) control the interface between the 10K-bit transmit storage buffer module (txmitbuffer.v) and the RS-232 transmitter module (txmit.v).

3.2.3 Forward Error Correction: Convolutional Encoder & 2-bit Serializer

NOTE: This component was not used in the final hardware implementation of this senior design project. The accompanying Viterbi Decoder (version 7.0) LogiCore module would not function properly during testing (see Evaluation), so this operational Convolutional Encoder module was unfortunately discarded as well.

The *Convolutional Encoder* (version 7.0) LogiCore module from the Xilinx Core Generator was elected for implementing non-punctured (2, 1, 7) convolutional encoding in the modem. The next figure shows the block diagram for this module. The target code rate was ½, so a single-bit input is specified and a two-bit output is specified. The module would operate using a 1200 Hz clock. The module operates continuously, whether it is encoding valid data or not. Hence, no other control signals are required.

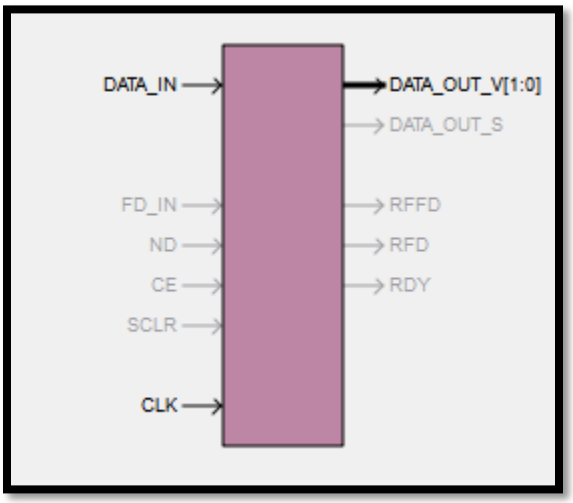

Figure 35. Block diagram of the Viterbi Decoder

At the positive edge of the clock, the encoder samples a bit present at the input pin. This bit is provided by the 10K-bit receive buffer (rcvbuffer.v). At the next positive edge of the clock, the encoder produces a two-bit result. This result is serialized (serializer.v) to create a 2400 bit/s data stream into the BPSK modulator.

3.2.4 Differential Encoding

After the first implementation, it was noticed that a random phase error was picked up by the Costas Loop's phase detectors. Unlike in simulation where the modulator and the demodulator are unconnected, the hardware implementation did not have this advantage and the phase angle between the modulated and the demodulated signals had random phase angles. This ambiguity would result in incorrect results from Costas Loop where the demodulated data would be the inverse of the actual modulated signal. The

addition of differential encoding ignored this 180° phase change. Differential encoding is based on a relationship between the previous and the current bit values. This results in an encoder with memory storage and with the following conditions:

- If the present bit is a 1, transmit a bit that is opposite to the previous bit
- If the present bit is a 0, transmit the value of the previous bit

This is accomplished with a temporary register and an exclusive-or operation, as shown in the difference equation in equation (26). The difference equation in Eq. 26 implies a bitwise exclusive or operation unto the current bit from the PC and the previously transmitted bit.

$$\boldsymbol{y[n] = x[n] \otimes y[n-1]} \tag{26}$$

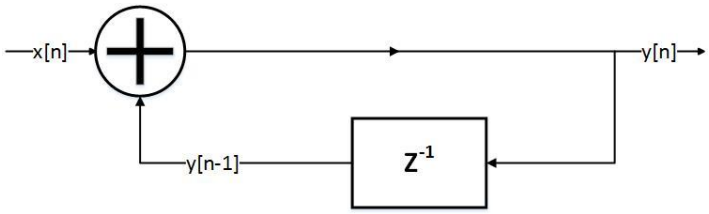

Figure 36. Functional block diagram of a Differential encoder

At the demodulator side, the data is decoded by performing the inverse operation of the encoder. This is accomplished by applying the exclusive-or to the receive bit and the previously received bit. Figure 37 illustrates the binary operation as a feed forward operation instead of the feedback like Figure 37.

$$\boldsymbol{y[n] = x[n] \otimes x[n-1]} \tag{27}$$

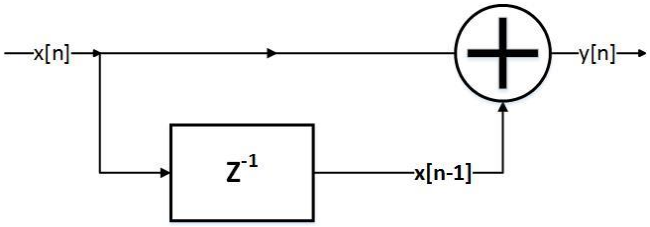

Figure 37. Functional block diagram of a Differential decoder

The operation of the differential encoder and decoder can be rendered as a Z-transfer function, following the operation done unto the input signal to the modem and output signal to the modem, the it can be shown that other that ignoring the phase of the signal, the differential encoder has no effect on the signal (Ngyen,252). The Z-transfer functions of the encoders and decoders are illustrated in equation (28) and equation (29) respectively.

$$\boldsymbol{Enc[z] = \frac{1}{1+z^{-1}}} \tag{28}$$

$$\boldsymbol{Dec[z] = 1 + z^{-1}} \tag{29}$$

Therefore as Figure 38 illustrates, the outcome renders to be $\frac{Dec[z]}{Enc[z]} = 1$.

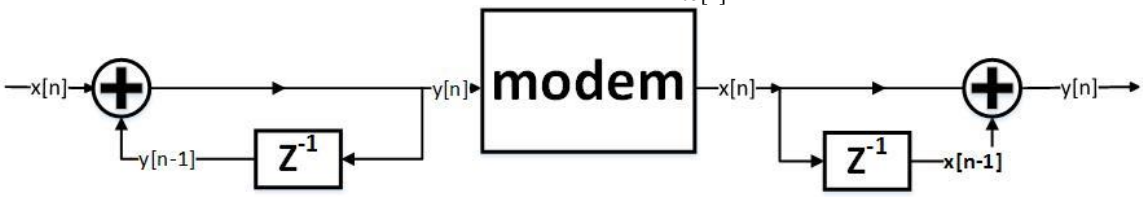

Figure 38. Implementation of the Differential encoder and decoder with respect to the BPSK modem.

3.2.4 FEC-BPSK Modulator

The BPSK modulator in the modem consists of a Direct Digital Synthesis Compiler version 4.0 (DDS Compiler) from Xilinx's intellectual properties. With incoming data bits coming from the *transmit buffer*, the DDS Compiler provides a carrier signal at 4800 Hz sampled at a rate of 2 MHz. Created in the digital domain, the precision of this sinusoidal waveform is dependent on the register size of the **output register** as well as the register size of the **phase width** register where its purpose is to look up the output magnitude with the use of a Look Up Table. A sampling rate of 2 MHz provides ≈ 416 samples per period, this number of samples assures that the carrier is well above the Nyquist Sampling Theorem and that the waveform can be processed in simulation. The output registers is set to define the amplitude of the BPSK waveform created in the digital realm, similarly, the amplitude of the waveform is also crucial for obtaining an appropriate step size from the modulator. For the modulator, the output width, (number of registers for the carrier signal) was designated as a signed bus of 12 registers, which results in a carrier signal with amplitude of $^{2^{12}}/_{2} - 1 = [-2047, 2048]$. With such amplitude, we expect our LUT to have 4098 values for the magnitude, therefore this will occupy $2{\times}4098$ registers for the look up tables. Just as the output magnitude and the sampling rate are important to the DDS Compiler, the phase width of the signal has as much importance to the output sine wave generated by the DDS Compiler. This parameter of the DDS Compiler dictates the frequency resolution of the sinusoidal wave generated where the frequency resolution is dependent on the sampling frequency and the phase width as follow:

$$\Delta \boldsymbol{f} = \frac{\boldsymbol{f_{clk}}}{\boldsymbol{2^{B_{\theta(n)}}}}, \tag{30}$$

Where $B_{\theta(n)}$ is defined as the phase width of the sinusoidal waveform, which the DDS Compiler uses in an accumulator for obtaining the output magnitudes.

For modulating the phase of the carrier signal, a sub-module to the *Modulator.v* module is created named *mixer.v*. This sub-module sees as input the 4800 Hz carrier signal from the DDS Compiler and the data bits from the *receive buffer* and later modulates the phase of the signal based on the values of the bits. BPSK modulation is done by reversing the polarity of the BPSK signal with respect to the data bits a bit 0° consist of a 180°, while a bit 1 is equal to a phase of 0°. This can easily be implemented as an "*if statement"* (or some conditional statement similar to Simulink) at a rate of 1200 Hz to synchronize the phase changes to the sampling clock and the bit stream. However the latency present in the DDS

Compiler and the sampling clock which is not a multiple of the carrier frequency slowly alters the phase value where the phase is needed to change (at 1200 Hz with a phase value of 0° *or* 180°). Therefore, the phase changes were controlled to only change the phase when the phase is precisely equal to 0° *or* 180°. Figure 39 illustrates the BPSK signal in the Frequency domain from the Electronics Explorer.

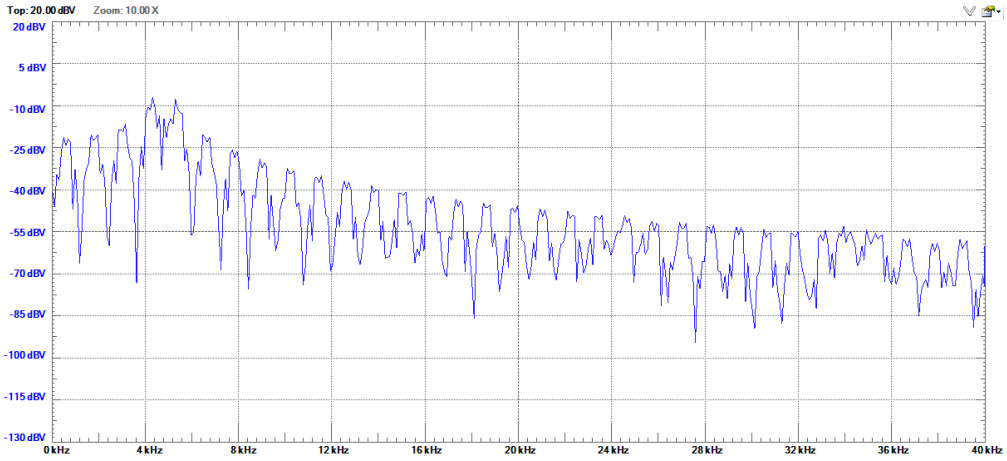

Figure 39. FFT of BPSK signal obtained using Electronics Explorer. Spectral analysis shows the suppressed carrier at 4800 Hz and first null at 6000 Hz (Suppressed carrier + bit rate).

The FFT shown in Figure 39 confirm the implementation of a 4800 Hz BPSK waveform as Silage describes in his literature, a BPSK waveform has suppressed carrier, or does not have a magnitude at that frequency and has null occurring at every integer multiple of the data rate starting from the carrier frequency. These are shown in Figure 39.

3.2.5 AWGN Channel

Note: Due to limitations with the AWGN core, this module was not implemented in the final design.

For testing the performance of the modem, Additive White Gaussian Noise is a sufficient and a must for BER evaluation. Since AWGN is defined to be a signal that is completely independent from other elements and with a mean of zero, it could be a complicated task to implement and with the AWGN IP Core discontinued by its owner, Xilinx, we turned to an AWGN module provided by Digi-Key. The *LFSR_Plus.v* module uses two Linear Feedback Shift Registers, the data is clocked by a non-uniform clock period which either enables the feedback value to add to the shift registers, while the last LFSR is used to compute an XOR equation between each of the data bits. The results are then stored in four temporary register which are shifted at the sampling rate, finally, the signal out of the module is the addition of the four registers. The Figure 40 shows the two Linear Shift Registers with the temporary registers summed up together, LFSR_1 provides a non-uniform clock that or a random enable which control the feedback of the LFSR_2, the output of the LFSR_2 is sent to first temporary registers which shifts its value down to second temporary register which passes as well and this process is executed all the way to the fourth register. The values of the registers are added to create the *g_noise* or the AWGN.

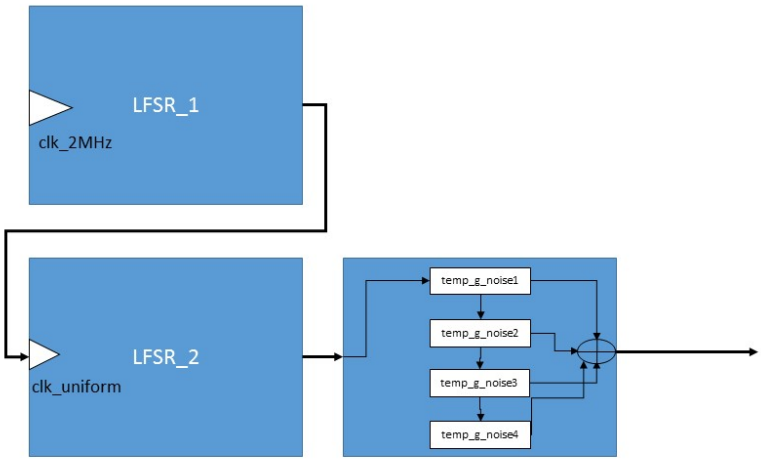

Figure 40. Block diagram of the LFSR module provided by Digi-Key

A test bench set up for the *LFRS_plus.v* was configured for testing the operation of the module. Figure 41 shows the output outcome of the LFSR module with 12 bits. Using Matlab, it was confirmed that the mean was equal to zero and that the distribution was indeed Gaussian as shown in Figure 41. Finding the variance, led to computing the Signal to Noise Ratio (SNR).

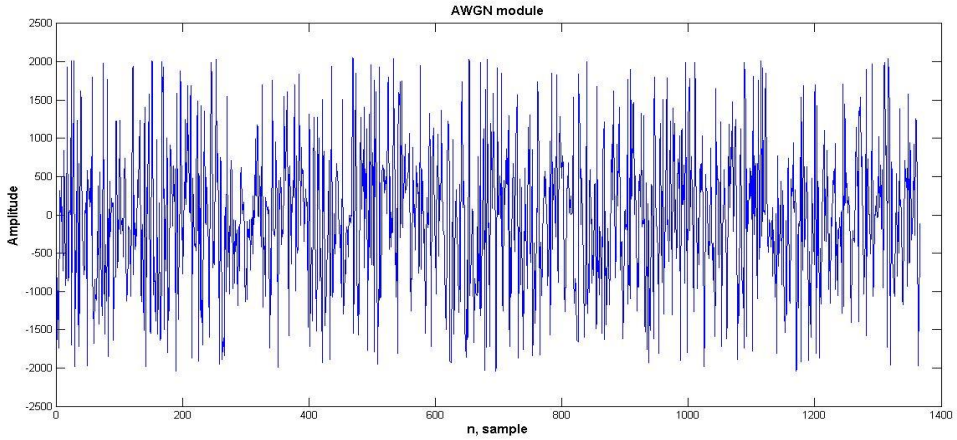

Figure 41 Results of the AWGN generator in simulation using Isim

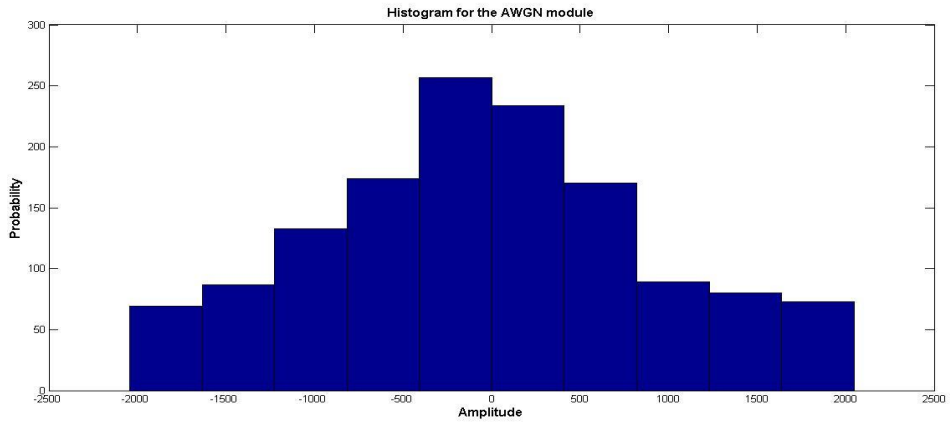

Figure 42. Gaussian noise obtained using the LFSR_plus.v module using ISIM (top); Histogram of the generated AWGN (bottom)

Outputting the Gaussian noise to the DAC, yielded an unexpected histogram. While the signal generated seemed to be Gaussian, the histogram obtained by the Electronics Analyzer, with a voltage range of -1.65V - 1.65V did not verify the results of Figure 42. With the distribution not completely Gaussian, this led to a recalculation of the SNR using the actual analog values of the AWGN signal.

3.2.6 FEC-BPSK Demodulator: Carrier, Timing, Data Recovery & Soft-decision Encoding

The implementation of the Costas Loop in FPGA follows from a direct translation of the Simulink Model in section 3.1.4. It uses a controller-datapath architecture and consists of the following modules:

1. Costas_Loop.v
2. Clock.v
3. CostasControl.v
4. Multiply.v
5. ArmFilter.v
6. PhaseDetector.v
7. LoopFilter.v
8. NCO.v

Costas_Loop.v is the top level wrapper, *Clock.v* generates the 2 MHz sampling clock, and *CostasControl.v* controls the enabling and disabling of Logicore blocks. The datapath consists of *Multiply.v*, *ArmFilter.v*, *PhaseDetector.v*, *LoopFilter*.v and *NCO.v*.

The module *Multiply.v* wraps two instantiations of Xilinx's Multiplier Logicore blocks. Each multiplier instantiation is optimized for speed with 12 bit inputs and 24 bit outputs. One instantiation multiplies the 12 bit received signal with the 12 bit I waveform from the DDS and the other multiplies the received signal with the 12 bit Q waveform from the DDS.

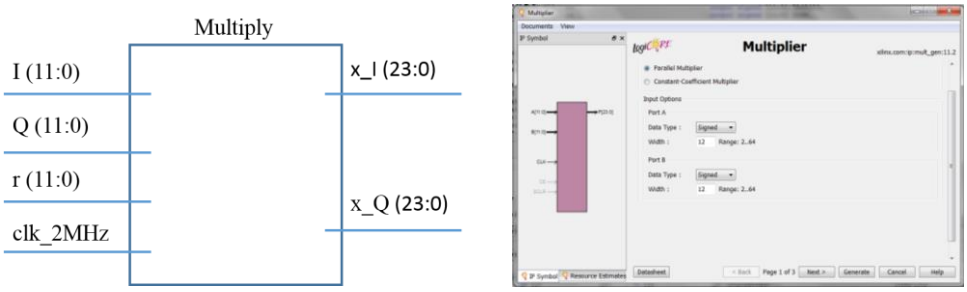

Figure 43. *Multiply.v* module that wraps two instantiations of Xilinx's Multiplier Logicore blocks.

Samples are multiplied at a 2 MHz rate and the 24 bit output feeds the *ArmFilter.v* module. Just like *Multiply.v*, *ArmFilter.v* wraps the I and Q arm filters into a single module. The I and Q arm filters were implemented using Xilinx's FIR Compiler Logicore blocks. The inputs to each arm filter are signed 24 bit samples at a rate of 2 MHz. The 100 MHz master clock drives each FIR compiler instantiation. A coefficient file was derived from Simulink's FDA tool that produces an identical magnitude and phase response in hardware. This is confirmed by comparing the magnitude response in Figure 44 with that of Figure 16's.

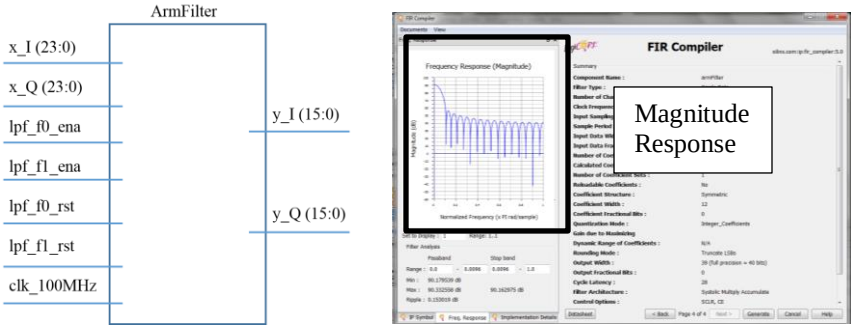

Figure 44. *ArmFilter.v* module that wraps two instantiations of Xilinx's FIR Compiler Logicore blocks.

The arm filter coefficient structures are symmetric and implemented as 12 bit unsigned integers. Full precision outputs require 40 bit signed outputs, but were truncated to 16 bits in accordance with the Costas Loop Specifications in Section 3.1.4. Each arm filter was also configured to have an enable and reset signal for control of the Costas Loop. The latency of each filter is 28 cycles.

The outputs of *ArmFilter.v* are wired to *PhaseDetector.v* which wraps a single Multiplier Logicore block. The inputs to the phase detector multiplier are 16 bit signed samples at a 2 MHz rate. The 32 bit signed output samples drive *LoopFilter.v*.

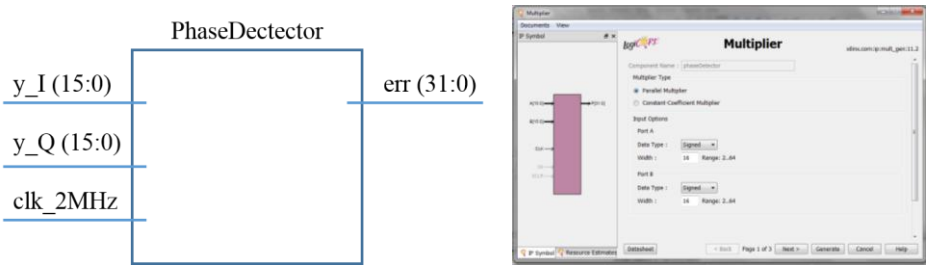

Figure 45. *PhaseDectector.v* module that wraps a single Multiplier Logicore block.

Unlike the multipliers and filters which were implemented using Logicore blocks, the loop filter was written in Verilog and implements a discrete time PI controller. The discrete time PI controller is implemented as a recursive IIR filter that replaces the multipliers in the difference equation with Verilog's arithmetic shift operator. This reduces the amount of DSP48 usages. The coefficients of the PI controller are set according to Table 4 in Section 3.1.4. The outputs of the loop filter is a 32 bit signed phase error the drives the DDS in *NCO*.v to the correct phase and frequency of the received signal.

The inputs to *NCO.v* are the 32 bit signed adjusted error samples from the loop filter, an enable, and a reset. A 2 MHz clock determines the output sampling rate. The output phase width is 12 bits while the phase increment width is 32 bits. This results in a spurious free dynamic range of 72 dB and a frequency resolution of 0.00047 Hz. This block is also optimized for speed and has a 2 cycle latency.

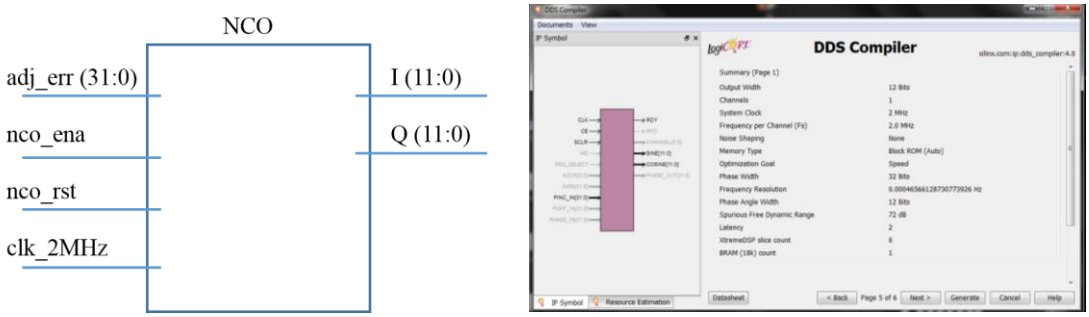

Figure 46. *NCO.v* module that wraps a single DDS Logicore block. The DDS adjust the phase and frequency of it's I/Q outputs to match that of the received signals.

It is important to note that the *adj_err* input of the DDS Compiler is the sum of the phase increment value that dictates the DDS center frequency and the phase error from the loop filter. The reason for this is simple, when the loop filter error is zero, the output frequency of the DDS will match that of the received signals. However, this is not well documented in literature. The phase increment value used in our design was computed in section 3.1.4 and listed in Table 4. Note that in Table 4, the phase increment value is referred to as the "VCO sensitivity gain." This illustrates one of the differences between simulation and implementation that is not well documented.

The final module that completes the hardware implementation of the Costas Loop is *CostasControl.v*. This module uses a simple FSM to sequentially enable all Logicore blocks after initialization. After initialization, the Costas Loop runs continuously until *rst* is set which synchronously resets all Logicore

blocks including the PI controller in *LoopFilter.v*. Figure 47 depicts the complete FPGA implementation of the Costas Loop.

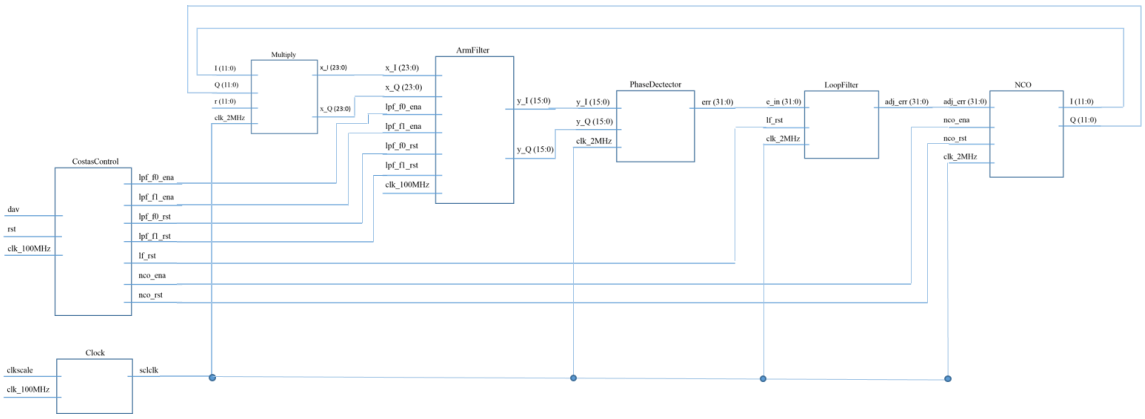

Figure 47. Complete FPGA implementation of the Costas Loop carrier recovery circuit.

Early Late Gate Hardware Implementation

After demodulating the BPSK signal, the NRZ signal is recovered from the Early Late Timing Recovery Circuit. The implementation of the timing recovery circuit is achieved in a similar manner as the one in Simulink and can be summarized by Figure 48. Two branches are implemented to recover the clock, a Late branch and an Early branch to recover the clock by subtracting the energy accumulated over a period of one bit time. The modules used in the Early Late gate module, are implemented to have the same function as the Simulink model. The integrator module *integrator.v* is coded as an accumulator operating at the 2 MHz, which is sampled by the positive edge of *clk*. The integrator module will then accumulate the demod signal and "dump" its value at the positive edge of the *clk_out* signal or the delayed version of the *clk_out* signal in the case of the late branch.

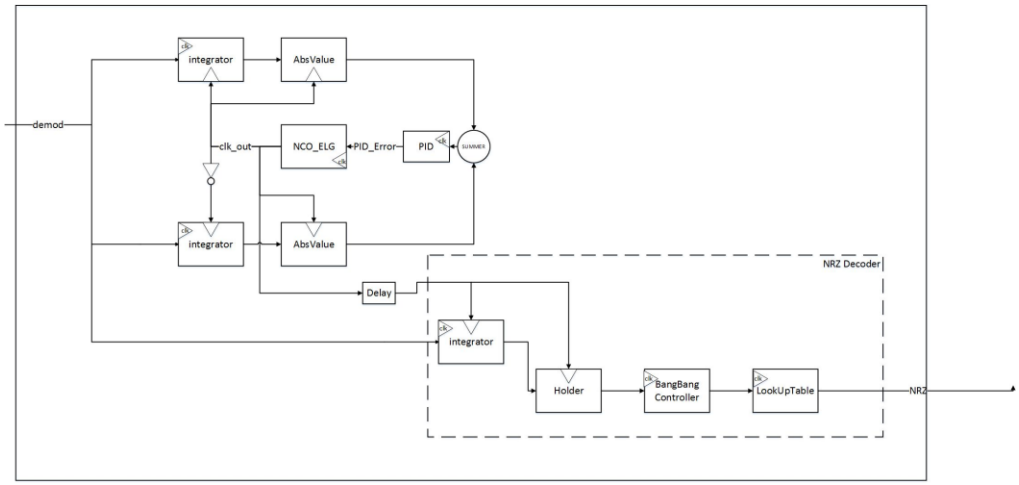

Figure 48. Early Late Gate block diagram in Verilog.

Figure 49 shows the output of the integrate module. ISIM and Matlab were used to illustrate the designed outcomes of the demodulator for the following figures. At the AbsValue modules, the absolute value of the input signal is taken and using the *clk_out* signal is sampled at a rate of 1200 Hz (which inherently samples & holds the value) and can be observed in Figure 50. In the SUMMER block, the Late branch is subtracted from the Early branch and then drives the PID controller. The PID controller is designed using the digital IIR filter with no more than three coefficients for the three parameters $x\,[n],\ x[n-1],\ y[n-1]$. With the difference equation shown in Eq. 31 the coefficients were obtained and shown in Table 6. As for the output of the PID filter, Figure 51 shows those results. Being a closed loop system, the controller needs to always adjust for the output clock, this is the explanation for the error signal going up and down in Figure 50.

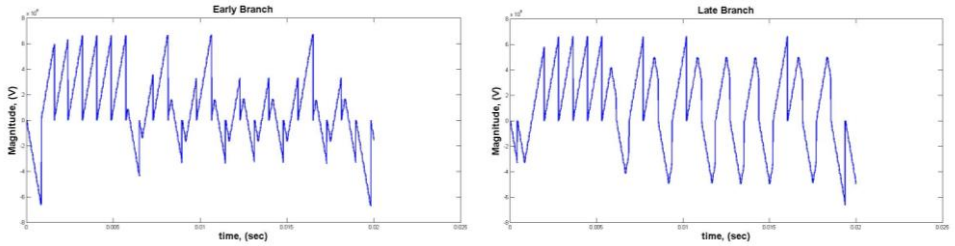

Figure 49. Early and Late integrator output (obtained using Matlab and ISIM).

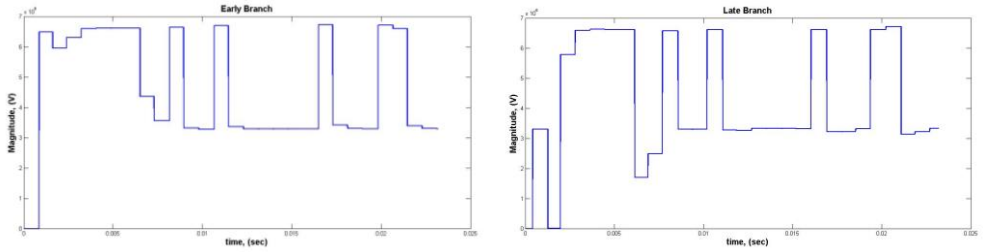

Figure 50. Early branch and the Late Branch after the sample & hold and absolute value (obtained using Matlab and ISIM)

$$\boldsymbol{y[n] = a \cdot x[n] + b \cdot x[n-1] + y[n-1]} \qquad (31)$$

a	b
1/15	1/15

Table 6. PID Filter coefficients for equation 31.

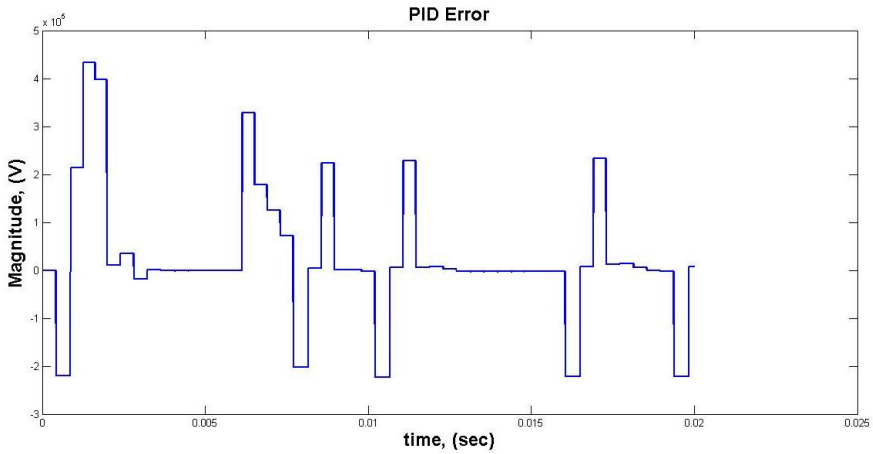

Figure 51. PID output, regulating the error signal to drive the NCO module (obtained using Matlab and ISIM)

The last component of the Early Late Gate is the NCO, which is used to "drive" the clock signal and completing the loop with the other elements. The adjusted error coming from the PID is the input of the port p_inc (the Phase Increment) of the DDS Compiler which is a sub-module to the NCO_ELG module. The phase increment port is similar to the *phase in* (input port) port in the Simulink block. The p_inc port is defined as a 32 bit signal, the p_inc is connected to the PID error and the quiescent frequency which is equal to 2400 Hz is converted into bits using equation (30) and is found to be equal to 2576980

The output signal is defined to be eight bits and is sent to the BangBangController which an on-off controller outputs a zero for any input less than zero and a one for any input greater than zero. From this set-up, the clock is expected to be extracted and provide the modules at the Early and Late branch a data clock, finally the output clock is shown in Figure 52.

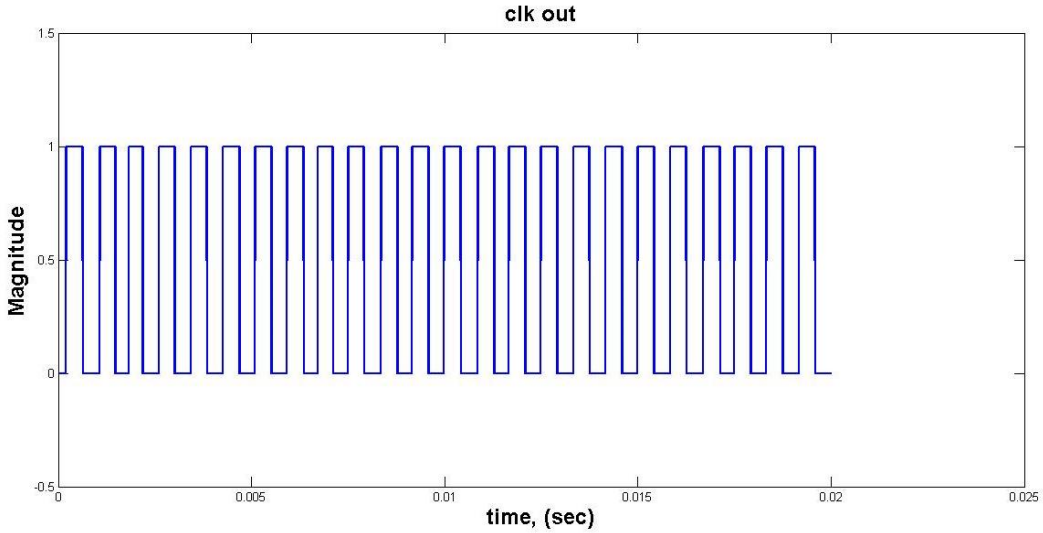

Figure 52. Recovered clock from the Early Late Gate (obtained using Matlab and ISIM)

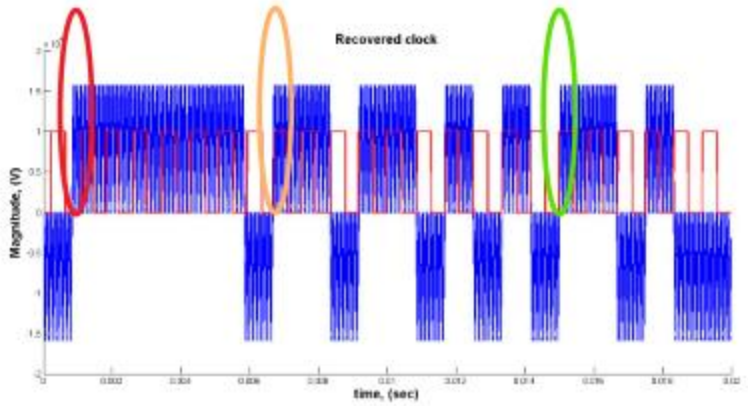

Figure 53. Recovered clock with demodulated data from the Costas Loop (obtained using Matlab and ISIM)

Figure 53 shows the recovered clock juxtaposed to the demodulated data from the Costas Loop. The rings encircle the data transition, from the red transition, the clock and the demodulated signal are not synchronized and as time progresses, and the clock signal becomes synchronized to the demodulated data.

Completely decoding the signal implies using the recovered clock signal shown in Figure 52, this clock signal is used for calculating the energy of the demodulated signal (from Costas Loop) and doing a hard-decision on whether the demodulated contains a bit 1 or a bit 0, this decision is essentially done within the *LookUpTable.v* module.

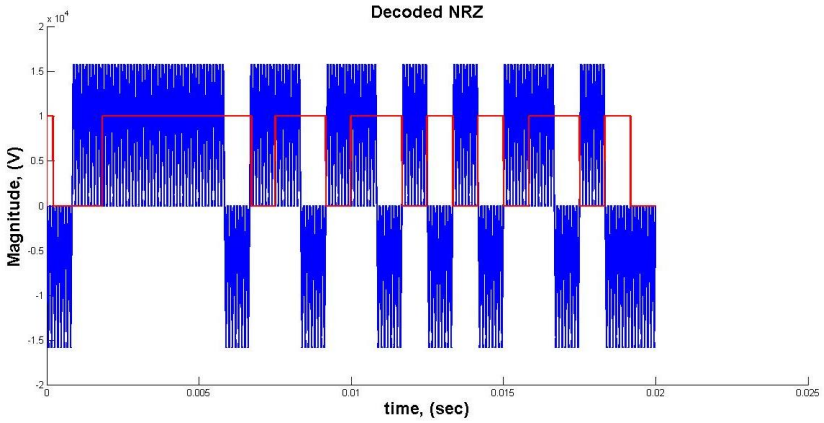

Figure 54. Decoded NRZ juxtaposed to the demodulated data from the Costas Loop (obtained using Matlab and ISIM)

Finally Figure 54 features the NRZ data unprocessed by the NRZ decoder and the decoded NRZ signal.

3.2.7 Forward Error Correction: Soft-decision Viterbi Decoding

NOTE: This component was not used in the final hardware implementation of this senior design project. The accompanying Viterbi Decoder (version 7.0) LogiCore module would not function properly during testing (see Evaluation).

The *Viterbi Decoder* (version 7.0) LogiCore module from the Xilinx Core Generator was elected for implementing 3-bit soft-decision, non-punctured (2, 1, 7) Viterbi decoding in the modem. Figure 55

shows the block diagram for this module. The target code rate is ½, so two three-bit inputs are specified and a single-bit output is specified. The module operates using a 1200 Hz clock. The module operates continuously, whether it is encoding valid data or not. By feeding in seven strong zeroes prior to the first valid data input (which is the first in a steam of valid data), trellis construction will begin in state zero (Francis, 2010). Hence, no other control signals like clock enable (CE) or new data (ND) are required for proper decoding.

At the positive edge of the clock, the decoder samples two three-bit soft-decision data present at the input pins. These two three-bit soft-decision results are provided by the soft-decision BPSK demodulator. The decoder (parallel core, without reduced latency option) produces its first valid data output after a delay specified by the following equation (Xilinx, 2011):

$$delay \cong 4{\times}traceback\ length + constraint\ length + output\ rate \cong 4{\times}42 + 7 + 2 \cong 177\ ms$$

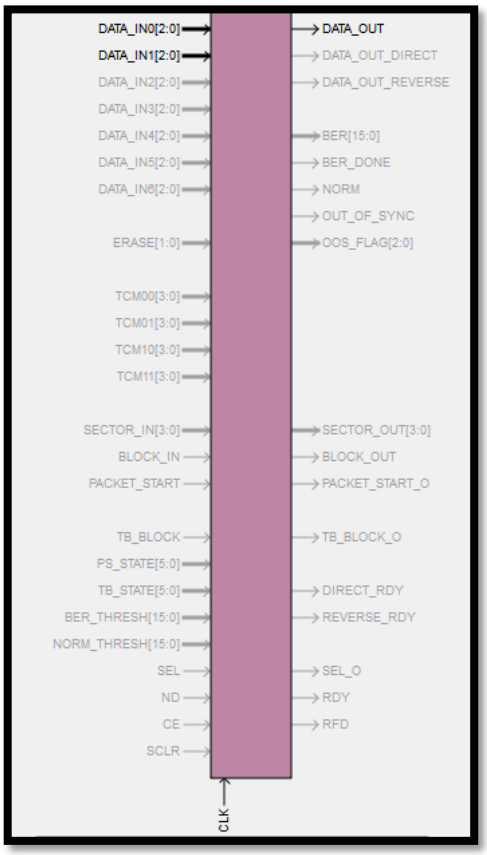

Figure 55. Block diagram of the Viterbi Decoder

4. EVALUATION

4.1. Modulation\ Demodulation

The modem abides to the amateur radio satellites requirements of transmitting at a rate of 1200 bps to ground stations using BPSK modulation. The modem maintains the required specification of transmitting signals with a carrier wave of 4800 Hz. The modulated signal is obtained using the Xilinx DDS Compiler which enhances the DSP capacities of the modem. Figure 56 illustrates the output BPSK signal using an analog oscilloscope. This signal contains repeated bits of 010101...

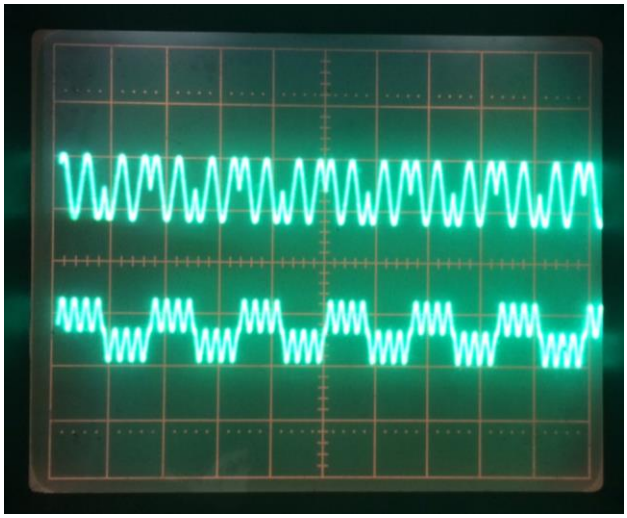

Figure 56. Modulated BPSK signal out of the modem

The received BPSK signal at the receiver station is demodulated coherently with the modem's Costas Loop and is juxtaposed to its modulated signal in Figure 57, a delay is picked up by the timing recovery circuit due to its integrators. The demodulated data is done using a second order type one Costas Loop, which means that the modem is able to track both phase and frequency step changes. Specifically, the lock range of the Costas loop is $\pm 200\ Hz$ from its center frequency of 4800 Hz. While the Costas Loop can safely track frequency offsets, it is unable to adjust to an unsynchronized BPSK signal with an offset angle greater than 90°. The solution to this well-known problem was the addition of a differential encoder which removed the phase ambiguity when decoding the NRZ signal. The results of the Early Late gate are shown in Figure 57 and Figure 58.

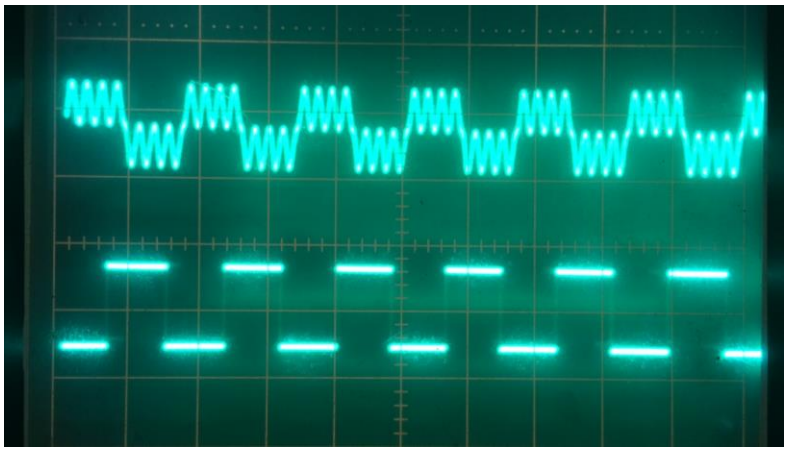

Figure 57. Costas Loop output (top) with NRZ signal (bottom)

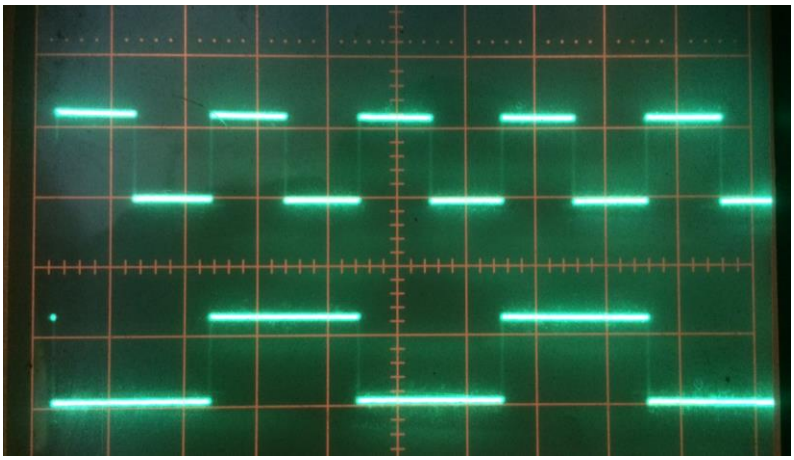

Figure 58. Clock signal (top) with NRZ signal (bottom).

4.2. Forward Error Correction

Forward error correction was unable to be achieved in the final rendering of this senior design project. We managed to get convolutional encoding and Viterbi decoding working properly in behavioral simulation, but we could not achieve the same in FPGA hardware. The next two figures show the behavioral simulation and hardware implementation, respectively. The behavioral simulation was performed in ISim. The hardware implementation was analyzed using ChipScope Integrated Logic Analyzer. In short, the first figure contains three figures within it. The uppermost figure is a 200 ms time capture of simulated interaction between convolutional encoder IP core and Viterbi decoder IP core. In simulation, all data inputted to the convolutional encoder ('data_in') was recovered by the Viterbi decoder ('vitout'). The data input was 1250 8-bit ASCII 'A' characters. The Viterbi decoder output begins after the ~177 ms processing delay derived in section 3.2.7.

However, when this successful design was generated into a programming file and run on the FPGA hardware, the same successful results could not be achieved. The second figure shows these same signals being scoped in Chipscope. Despite similar conditions and data inputs (M4/M5/data_in) occurring in the exact same sequence, the output of the Viterbi decoder ('M4/M8/data_out') always was erroneous in

hardware. Actually, the output was not completely erroneous; there was a pattern to the error. See the third figure for the data that was received by the serial terminal via the Viterbi decoder. There is clearly a pattern. Hence, the Viterbi decoder tried to work properly, but some issue persisted in the IP core. We consulted Xilinx Answer records (specifically, AR# 22374), and the solutions did not help us. Hence, we terminated our use of the convolutional encoder and Viterbi decoder in this senior design project.

Figure 59: Simulated outcomes for the convolutional encoder- Viterbi Decoder using ISim

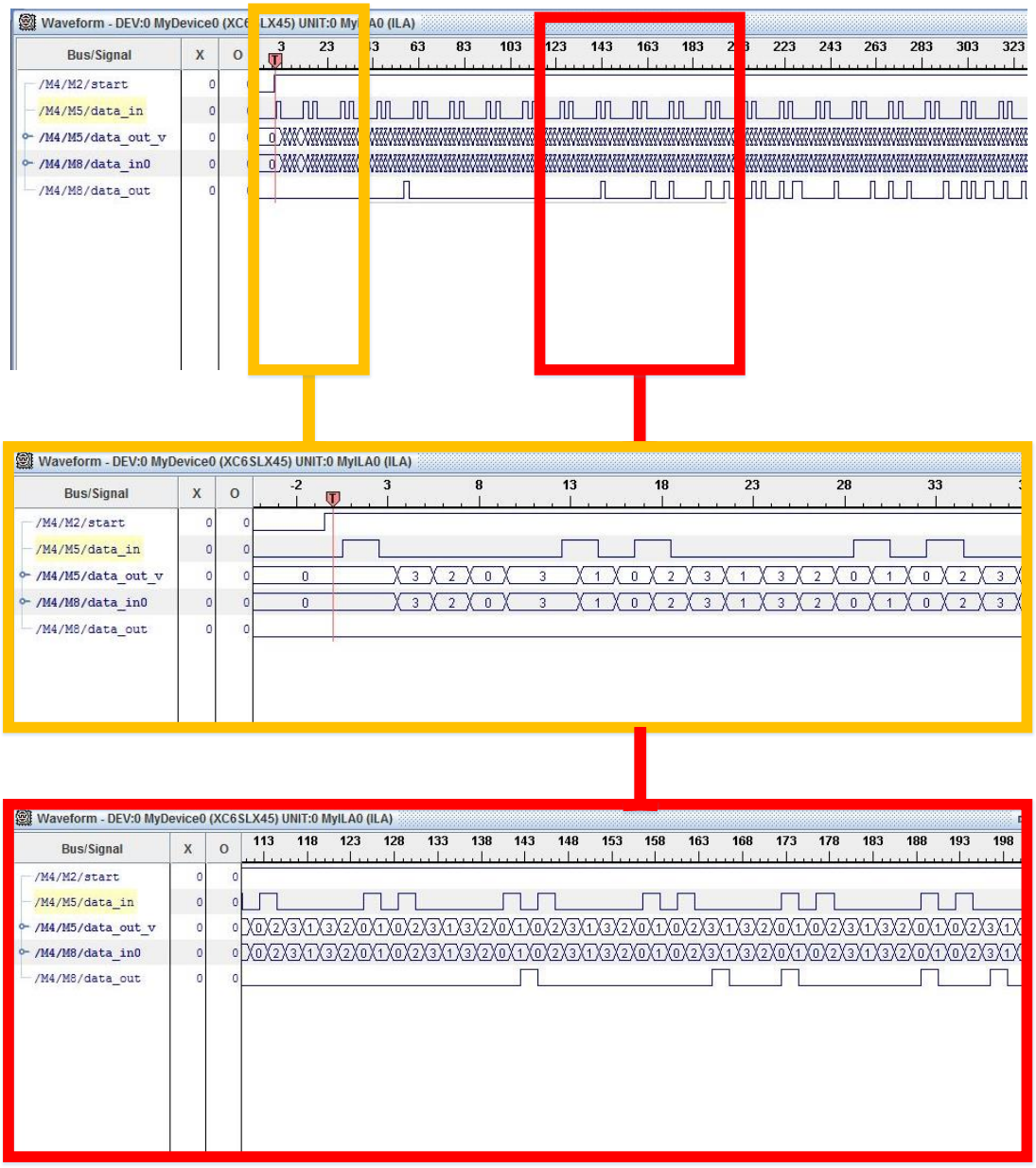

Figure 60 Convolutional encoder- Viterbi Decoder in the DUT analyzed using ChipScope Analyzer

Since we could not get the forward error correction working properly in our senior design project, a number of functional design constraints were not met in Table 1. However, the above figures does verify the modem can properly receive 1200 bit/s data stream, even though that wasn't a part of the functional design constraints.

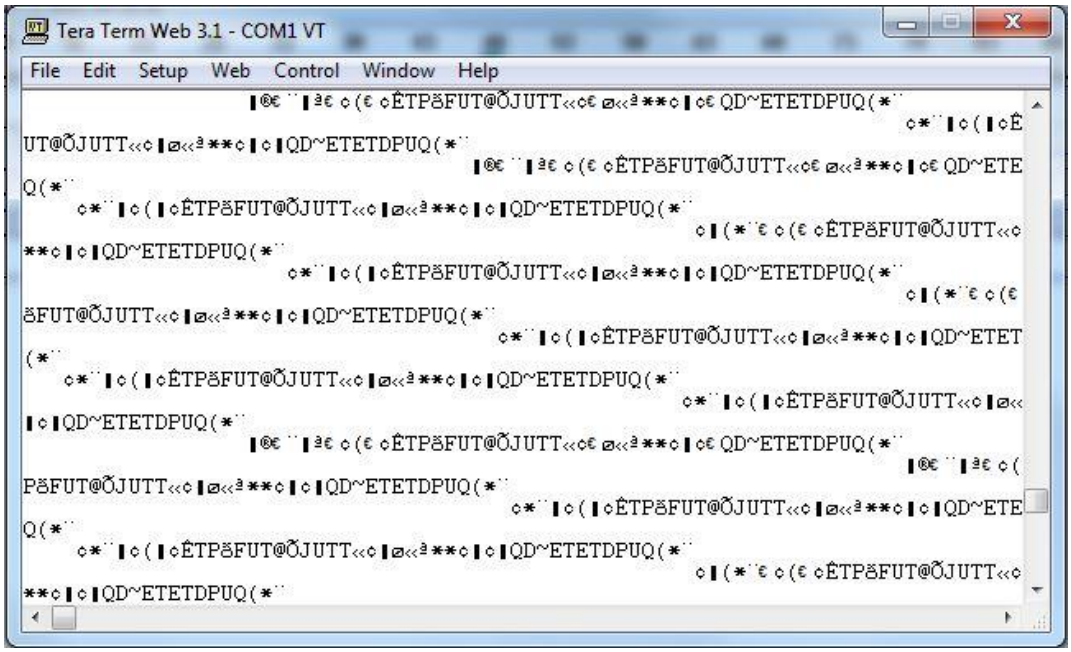

Figure 61: ASCII results obtained from transmitting characters from A-Z, a-z and 0-9 using the Viterbi Decoder

5. SUMMARY AND FUTURE WORK

This senior design project designed and implemented a 1200 b/sec BPSK modem in FPGA for amateur radio satellite telemetry. A 1200 b/sec data rate was chosen because the majority of operational AMSAT's operate at this rate. Our objective was to promote FEC in amateur radio satellite telemetry with addition of convolutional encoding and Viterbi decoding. However, issues with the Viterbi decoder IP core prevented us from successfully implementing FEC.

In senior design I, the modem's specifications and requirements were defined. This was accomplished by identifying the primary subsystems in the modulator and demodulator. This included the Costas Loop for carrier recovery and the Early-Late gate for clock and data recovery. After having identified the primary subsystems, the modem was designed and simulated from first principles using Matlab and Simulink.

In Senior Design II, the design of the modem transitioned from simulation to hardware by directly translating the Simulink model into Verilog code. The modem was implemented in a Spartan 6 FPGA and interfaced to a PC via RS232. The modem was evaluated by sending 10,000 bits (1250 ASCII characters) from the PC to the FPGA and computing the bit error rate. Under conditions with no noise, it was demonstrated that the modem had a BER of 0%. Further evaluation showed that the demodulator was capable of tracking both phase and frequency steps specified in the section 2.

Future work still includes adding FEC. Instead of using licensed IP cores such as the Convolutional Encoder and Viterbi Decoder, we want to use linear block codes. This allows the modem technology to be more accessible to the amateur radio enthusiast. After the addition of FEC using linear block codes, the modem needs to be integrated with an analog front end and a terminal node controller (TNC) to evaluate the modem using real world telemetry signals. The TNC is required to decode the AX.26 packets utilized in AMSAT telemetry. In addition this will allow us to do a thorough BER evaluation under real world noisy conditions

6. ACKNOWLEDGEMENTS

We would like first to thank Dr. Dennis Silage for his guidance and support throughout the year, which helped design and finalize our project onto the FPGA. The System on Chip Design Lab provided with all the design and testing tools required to accomplish this project and we honestly appreciate Dr. Silage allowing us to establish our workbench among his graduate student. We also would like to give a special thanks to the College of Engineering and especially the Electrical Engineering Department which made a tremendous job uniting the Programmable Communications Group for this wonderful project idea.

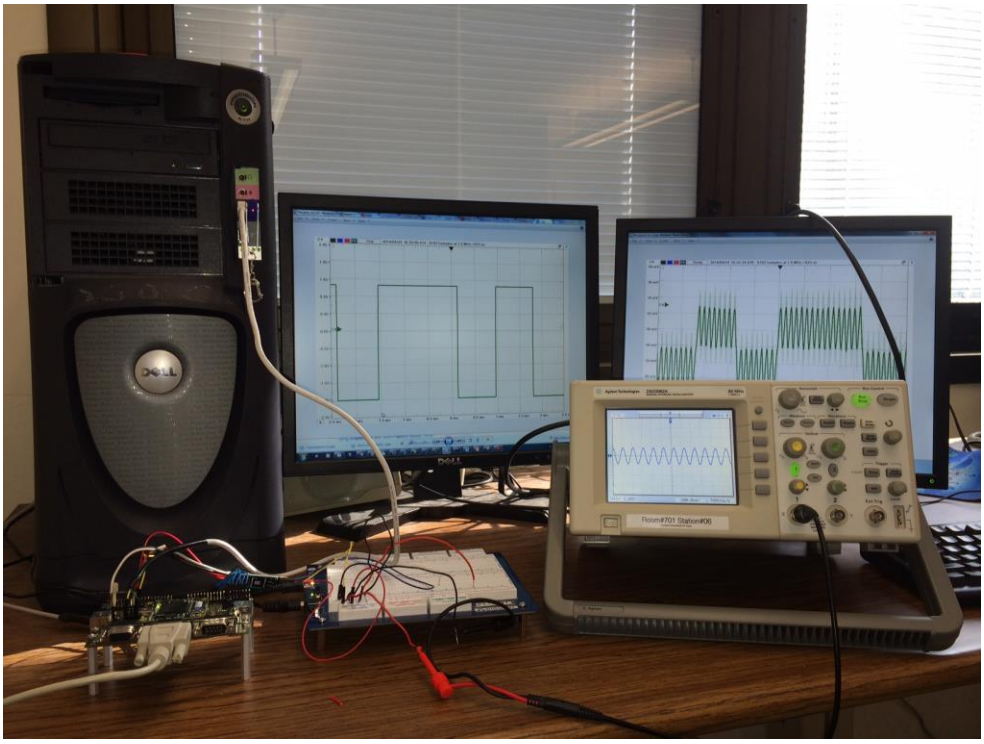

Figure 62: Programmable Communications Group's Workbench in the System on Chip Design Laboratory

7. REFERENCES

Alminde, L., Bisgaard, M., Vinther, D., Viscor, T., & Ostergaard, K. Z. (2002). *Robustness of radio link between AAU-cubesat and ground station.* Unpublished manuscript.

Capitaine, T., Barrandon, L., Bourny, V., Senlis, J., Le Mortellec, A., Astier, R., et al. (2010). Robust satellite AX25 frames demodulation. Paper presented at the *The Small Satellites Systems and Services (4S) Symposium, , 31.*

Crawford, J. A. (2007). *Advanced phase-lock techniques*. Norwood, Mass.: Artech House.

Davis, J. (2010). Amateur radio's lost future. Message posted to http://www.amateurradio.com/ham-radios-lost-future/

de Milliano, M., & Verhoeven, C. (2010). Towards the next generation of nanosatellite communication systems. *Acta Astronautica, 66*(9–10), 1425-1433. doi:http://dx.doi.org.libproxy.temple.edu/10.1016/j.actaastro.2009.10.034

Feigin, J. (2002, January 1). Practical Costas Loop Design. *RF Signal Processing*, *NA*, 20-26. Retrieved September 14, 2013, from http://defenseelectronicsmag.com/site-files/defenseelectronicsmag.com/files/archive/rfdesign.com/images/archive/0102Feigin20.pdf

Goode, S. (1984). BER performance of TAPR TNC modem. *Packet Status Register,* (11), 14-15.

Hsiao, F., Liu, H., & Hsieh, S. (2000). Using forward error correction technique for microsatellite data broadcasting system. *Acta Astronautica, 46*(2–6), 203-211. doi:http://dx.doi.org.libproxy.temple.edu/10.1016/S0094-5765(99)00224-6

Judd, D. (1996). Data synchronization simulation using the MATHWORKS Communications Toolbox. *1996 IEEE International Conference on Communications*, *2*, 706-710.

Karn, P. (1994). *Toward new link layer protocols.* Retrieved 11/01, 2013, from http://www.qsl.net/n9zia/newlinkpaper.html

Karn, P. (2011). *The BPSK1000 telemetry modem for ArriSSat-1.* Retrieved 11/01, 2013, from http://www.ka9q.net/bpsk1000.html

Lindsey, W., & Simon, M. K. (1977). Detection of digital FSK and PSK using a first-order phase-locked loop. *Communications, IEEE Transactions on, 25*(2), 200-214. doi:10.1109/TCOM.1977.1093804

Magliacane, J. (1997). Digital OSCAR Communications. *Digital OSCAR Communications*. Retrieved December 2, 2013, from http://www.amsat.org/amsat/articles/kd2bd/Digital/

Murphy, J., Chow, E., & Markley, R. (1994). ATM service-based selective retransmission over DSN satellite links. Paper presented at the *15th AIAA International Communication Satellite Systems Conference, San Diego, USA, , 350.*

Nguyen, H., & Shwedyk, E. (2009). *A first course in digital communications* (1st ed.). Cambridge, UK: Cambridge University Press.

Proakis, J. G. (2008). *Digital communications* (5th ed.). Boston: McGraw-Hill.

Rao, K. H. S., Jamadagni, K. S., Von Allmen, L. A., & Shah, A. V. (1990). All-digital pseudo-coherent (PC) FSK modems. Paper presented at the *Digital Communications, 1990. Electronic Circuits and Systems for Communications. Proceedings, 1990 International Zurich Seminar on,* 463-473. doi:10.1109/DIGCOM.1990.129390

Roelofs, E. (1987). a fsk modem for the map protocol. *Report of the Practical,*

Silage, D. (2009). *Digital communication systems using MATLABÂ® and SimulinkÂ®*. Gilroy, CA: Bookstand Pub..

Sklar, B. (2001). *Digital communications: Fundamentals and applications* (2nd ed.). New Jersey: Prentice Hall.

Viswanathan, M. (2013). In Mathuranathan V. (Ed.), *Simulation of digital communication systems using matlab* (2nd ed.). GaussianWaves: Viswanathan, Mathuranathan;.

Wallio, R. *AX.25 packet radio AFSK on FM Bit, packet and data set error rates.* Retrieved 11/07, 2013, from http://showcase.netins.net/web/wallio/BER_Packetradiobiterrorrate.html

Zicari, P., Corsonello, P., & Perri, S. (2008). A high flexible Early-Late Gate bit synchronizer in FPGA-based software defined radios. *4th European Conference on Circuits and Systems for Communication*, *NA*, 252-255.