An FPGA Implementation of Carrier Phase and Symbol Timing Synchronization for 16-APSK

Jason Baxter

Submitted to the graduate degree program in Electrical Engineering and Computer Science Department and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of M.S. in Electrical Engineering.

Dr. Erik Perrins, Chairperson

Committee members

Dr. Taejoon Kim

Dr. Carl Leuschen

Date defended: August 19, 2020

The Thesis Committee for Jason Baxter certifies

that this is the approved version of the following thesis :

An FPGA Implementation of Carrier Phase and Symbol Timing Synchronization for 16-APSK

Dr. Erik Perrins, Chairperson

Date approved: August 19, 2020

Abstract

Proper synchronization between a transmitter and receiver, in terms of carrier phase and symbol timing, is critical for reliable communication. Carrier phase synchronization is related to the frequency translation hardware, where perfect synchronization means that the local oscillators of the transmitter's upconverter and receiver's downconverter are aligned in phase and frequency. Timing synchronization is related to the analog-to-digital converter in the receiver, where perfect synchronization means that samples of the received signal are taken at transmitted symbol times. Perfect synchronization is unlikely in practical systems for a number of reasons, including hardware limitations and the independence of the transmitter and receiver. This thesis explores an FPGA implementation of a PLL-based carrier phase and symbol timing synchronization subsystem as part of a 16-APSK aeronautical telemetry receiver. The theory behind this subsystem is presented, and the hardware implementation of each component is described. Results demonstrate successful demodulation of a test signal, and system performance is shown to be comparable to double-precision floating point simulations in terms of error vector magnitude, synchronization lock time, and BER.

Acknowledgements

I will forever be grateful to Dr. Perrins for inviting me to join his project and pushing me to attend graduate school. I was extremely fortunate to be a part of this group with Dan and Ed, whose patience allowed me to learn so much over the last three years.

To Sarah, thank you for sitting with me at Nichols for many, many hours and nodding while I talked out my problems on bits and clock frequencies. I'm happy that I've been able to celebrate every small milestone along the way with you.

To my parents, sisters, Altus, and all of my friends, thank you for your understanding and support. I hope you all read the entire thesis.

Contents

List of Figures

List of Tables

Chapter 1

Background and Introduction

1.1 Context of This Work

The work in this thesis is part of a project investigating the advantage of amplitude and phase shift keying (APSK) over shaped-offset quadrature phase-shift keying (SOQPSK-TG) for aeronautical telemetry in terms of increased spectral efficiency. SOQPSK-TG, a modulation scheme traditionally used in telemetry, suffers a poor spectral efficiency due to its low number of bits per symbol. This has prompted the exploration of APSK to increase efficiency by encoding information in both the magnitude and phase of the transmitted signal. A main goal of the project is to show that the telemetry standard for minimum adjacent channel spacing can be reduced due to both the inherently smaller bandwidth of an APSK signal and an increased resistance to adjacent channel interference.

However, a significant tradeoff between a continuous phase modulation (CPM) scheme, like SOQPSK, and APSK is APSK's increased sensitivity to nonlinearities of a transmitter's RF power amplifier (PA). Unlike CPM systems, in which the PA can be driven into full saturation without regard to gain distortion, a system with a varying envelope like APSK could experience bit errors due to the PA's nonlinear behavior as it saturates. This requires the PA to be operated in its linear gain region, which decreases the PA's power efficiency in addition to reducing the system's signal-to-noise ratio (SNR). Some of the SNR lost by backing off the PA power level can be recovered by introducing forward error correction (FEC) coding. With FEC coding gain, a signal with a

lower SNR can exhibit the same bit error rate (BER) performance as a signal with a higher SNR. This project employs low density parity check (LDPC) FEC codes to achieve this performance boost. LDPC codes approach channel capacity, and their decoding algorithm can be efficiently implemented with parallel processing.

As part of this project, this thesis focuses on the digital signal processing at the receiver. Specifically, this thesis describes an FPGA implementation of the receiver's carrier phase and symbol timing synchronization system. The next section provides a brief overview of why this system is necessary for successful demodulation.

1.2 Introduction to Synchronization

The job of the receiver in a digital communication system is to extract the original information signal from the modulated (i.e. upsampled and filtered) signal. Most discussions of demodulation techniques assume perfect synchronization in timing and phase between the transmitter and receiver. Under this assumption, the received signal can be demodulated by simply filtering and downsampling, and any errors would be due to the channel. However, practical synchronization issues arise when independent transmitter and receiver hardware is used for communication. In general, there are two offsets between the transmitter and receiver that affect synchronization: A phase offset and a timing offset.

The demodulated baseband signal will exhibit a phase offset if the oscillator used for downconversion at the receiver is not aligned in phase and frequency with the transmitter's upconversion oscillator. As a result of this phase offset, the receiver's samples will fall in between constellation points rather than lining up properly. If the phase offset is large enough to move the samples to a different decision region of the constellation, then decision errors will occur even in the absence of noise. This issue is discussed in further detail in Section 3.1.

A timing offset also produces errors even if no noise is present. A symbol timing offset implies

that the analog-to-digital converter (ADC) is not sampling the received analog signal at the ideal sampling times. For a signal that has been upsampled at rate N, every N^{th} sample would ideally map directly to a constellation point. However, if sampling is not initiated at the exact starting time of the signal, then every N^{th} sample will instead fall between the desired values, potentially resulting in decision errors. This issue can also be represented through a signal's eye diagram, where decisions are made at times before or after the eye diagram's maximum opening. This concept is discussed in further detail in Section 3.2.

Because of these timing and phase errors, a synchronization system that will recover the original timing and phase information is needed at the receiver. The work described in this thesis tracks these offsets and processes the received samples accordingly to allow for successful demodulation.

1.3 Organization of This Thesis

The receiver components around the synchronization system are introduced first, including the frequency translation blocks, the pulse shape, and the LDPC decoder. Next, a theoretical description of the synchronization system is given. Then, the hardware implementation of each synchronization block is described. Finally, results of the FPGA implementation are compared with simulations to confirm successful demodulation of a 16-APSK signal.

1.4 Notation

The notation throughout this thesis refers to signals within the synchronization system. The received signal is generally denoted r, with I and Q components x and y at complex baseband. The indexing of r changes as the signal moves through the receiver. The sample time is T, and the symbol time is T_s. Samples are indexed with n, and symbols are indexed with k. The I and Q components of the kth decision are denoted $\hat{a}_I(k)$ and $\hat{a}_Q(k)$. The upsampling factor of the transmitted data is N.

Chapter 2

Receiver System Description

The synchronization subsystem of a digital communications receiver is the focus of this work. This chapter provides a high-level description of the other important components of the receiver.

2.1 From RF to Complex Baseband

The first task in almost any communications system is to step the antenna's received RF signal down to the system's intermediate frequency (IF). There are several reasons for downconversion to IF: First, hardware that performs well at lower frequencies is more practical than hardware for high frequencies. Furthermore, converting to IF allows systems to be designed for a common frequency so that signals at different carrier frequencies can be processed by the same system. In this project, the C-band RF signal is downconverted to a commonly used IF of 70 MHz.

For digital communications, most processing, including synchronization, is performed at complex baseband. After being sampled by an analog-to-digital converter (ADC), the IF signal is downconverted to complex baseband using two mixers with a $\pi/2$ phase difference. Low-pass filters then remove high frequency components from the mixer output, resulting in the I and Q components of the signal. The RF to baseband translation is shown in Figure 2.1. The quadrature mixers are represented in Figure 2.1 by a single multiplication with the complex exponential. The downconversion from RF to IF is treated as a black box for the work in this thesis.

As detailed in [1], downconversion from IF to baseband can be achieved without arithmetic by

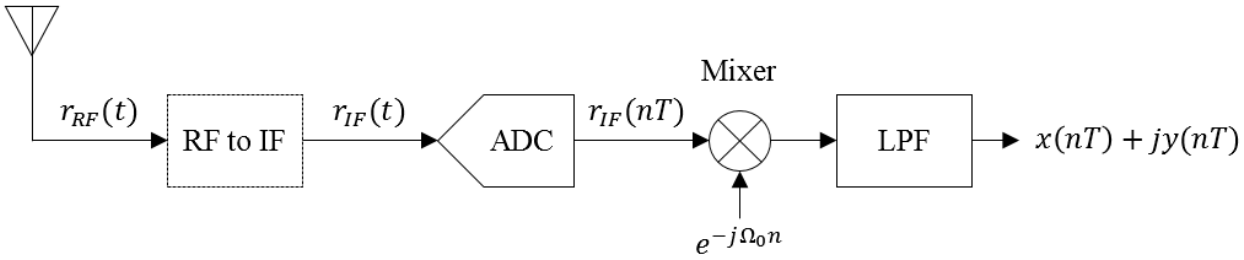

Figure 2.1: RF to Complex Baseband Downconversion

sampling the IF signal such that the sampled spectrum is centered at $\Omega_0 = \pi/2$. In general, digital frequency is defined as $\Omega_0 = 2\pi f_0/f_s$ radians per second, where f_s is the ADC sampling rate of the continuous-time signal centered at frequency f_0. Therefore, any sampling rate satisfying (2.1) can be chosen for the ADC, where k is an integer that comes from the periodicity of Ω_0.

$$kf_s \pm \frac{1}{4} f_s = f_0 \tag{2.1}$$

The sampling rate of the analog-to-digital converter (ADC) in this system was set to a commonly used frequency $f_s = 93\frac{1}{3}$ MHz. As previously stated, this sampling rate eliminates the need for the complex multiplications at the mixer shown in Figure 2.1. Because $\Omega_0 = \pi/2$, the complex exponential $e^{j\Omega_0 n}$ will be a repetition of the sequence $[1, j, -1, -j]$. Consequently, only a pattern of holds and sign reversals is required for mixing.

2.2 Pulse Shape

Many references will be made to the receiver's "matched filter" throughout this thesis. This filter is applied to the baseband signal before the signal is processed by the synchronization system. It is called a matched filter because the same pulse is used by the transmitter to filter the baseband signal before upconversion to RF. In general, for a receiver, matching the transmit pulse (with a time reversal) is the optimum pulse shape in terms of maximizing SNR, i.e. $p_{rx,opt} = p_{tx}(-t)$.

Pulse shaping is used in communications systems to limit the bandwidth and intersymbol interfer-

ence (ISI) of a signal. An ideal pulse shape is finite in the frequency domain (and therefore infinite in the time domain), and it satisfies the Nyquist no-ISI criterion [2]. A common pulse shape that meets these requirements is the square-root raised-cosine (SRRC) pulse. The discrete-time version of the SRRC pulse is given in (2.2), where $-\infty < n < \infty$ is an integer, $1/T$ is the sample rate, and $0 \leq \alpha \leq 1$ is a design parameter called the SRRC rolloff factor.

$$p(nT) = \frac{1}{\sqrt{N}} \frac{\sin\left(\pi(1-\alpha)\frac{n}{N}\right) + \frac{4\alpha n}{N}\cos\left(\pi(1+\alpha)\frac{n}{N}\right)}{\frac{\pi n}{N}\left[1-\left(\frac{4\alpha n}{N}\right)^2\right]} \tag{2.2}$$

Although the SRRC pulse eliminates ISI, it is not practical because of its infinite time duration. For implementation, the pulse must be truncated to a finite length such that it spans L symbols. The truncated pulse is no longer infinite in time, which means it is infinite in frequency and exhibits frequency side lobes. Additionally, the truncated pulse no longer satisfies the Nyquist no-ISI criterion.

Several parameters for SRRC pulse design can be seen in the discussion above: The rolloff factor α, the filter span L symbols, and a windowing function to attempt to minimize the frequency side lobes. The pulse used in this work was studied in [3] and is summarized in Table 2.1. The pulse is shown in the time domain in Figure 2.2 and in the frequency domain in Figure 2.3.

Table 2.1: SRRC Pulse Parameters

rolloff	0.4051
span	16 symbols
window	Kaiser
window shape factor	2.8299

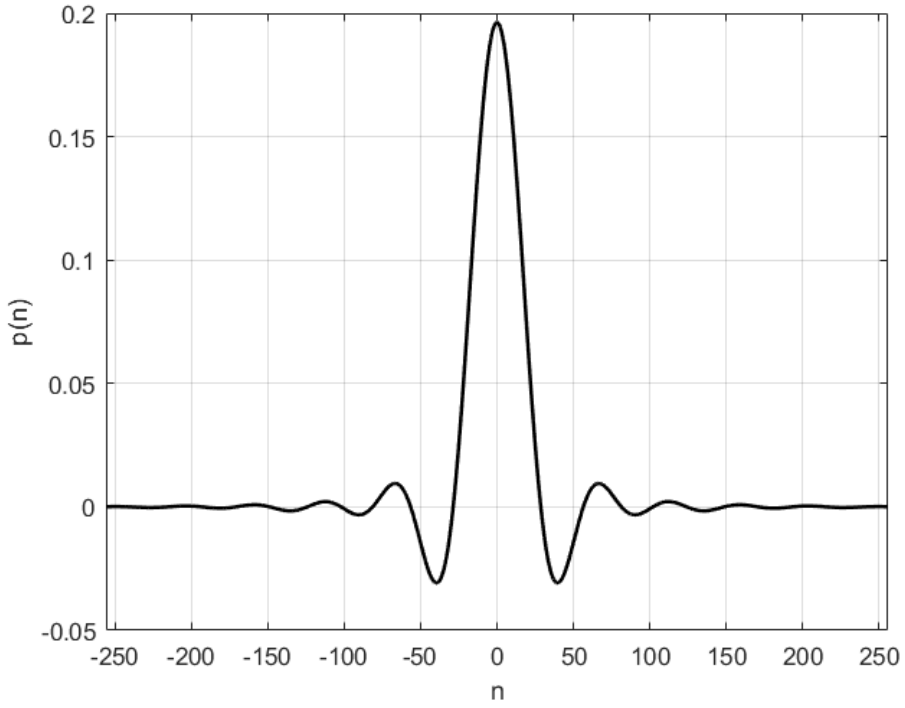

Figure 2.2: Windowed SRRC Pulse

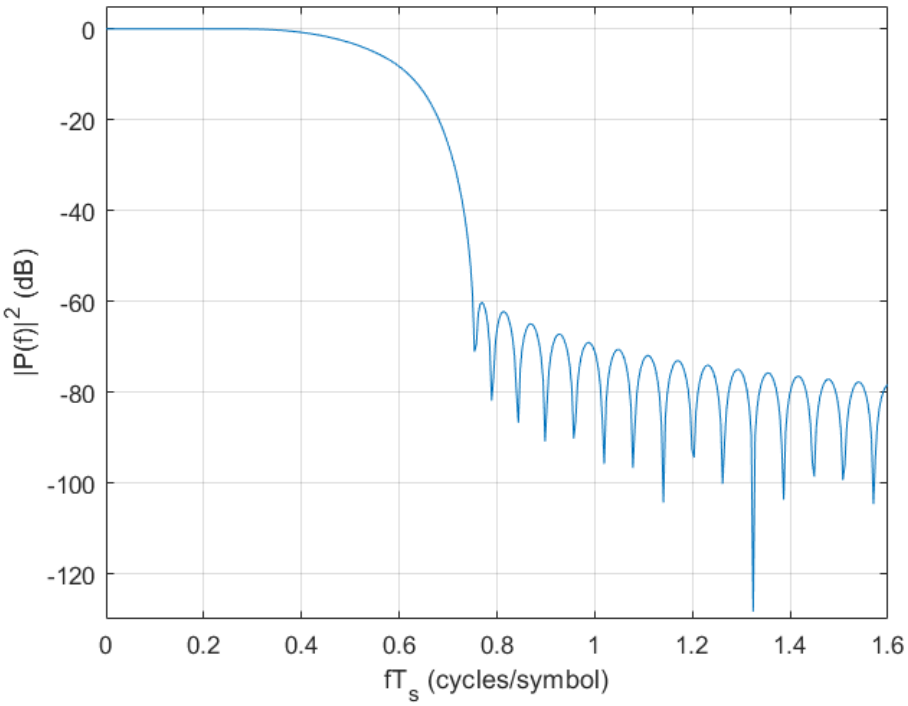

Figure 2.3: Windowed SRRC Pulse Power Spectrum

2.3 LDPC Decoder

The LDPC decoder input is significant for this work because the output of the FPGA is sent directly to the decoder, so it must be formatted correctly. The LDPC decoding algorithm is discussed briefly for completeness; however, it was treated as a black box for the work in this thesis.

2.3.1 Decoder Input

The LDPC decoding algorithm works on soft decisions. This means that the input to the decoder is the probability of each bit in a symbol given the demodulated vector. The basis behind this is that if, for example, the first bit of all of the decision regions around a vector is a 1, then there is a strong probability that the first bit of the symbol is a 1 even if the noise variance is high. However, if the vector falls near the boundary of two decision regions with different first bits, then the confidence in the first bit being a 1 or 0 is lower. After taking the log of this probability, it is known as a log-likelihood ratio (LLR). The LLR λ_i is shown in (2.3) for the ith bit b_i given the demodulated vector r.

$$\lambda_i = \ln\left(\frac{P(b_i = 0|r)}{P(b_i = 1|r)}\right) \tag{2.3}$$

Assuming the *a priori* probability of receiving a 1 is equal to the *a priori* probability of receiving a 0, (2.3) can be rewritten using Bayes' rule as shown in (2.4).

$$\lambda_i = \ln\left(\frac{P(r|b_i = 0)}{P(r|b_i = 1)}\right) \tag{2.4}$$

The work in [4] finds (2.4) as (2.5), where $\mathbf{a}$ is a constellation point and σ^2 is the noise variance of the received signal assuming an AWGN channel.

$$\lambda_i = \ln\left[\frac{\sum_{\mathbf{a}:b_i=0} \exp(\frac{\langle r,\mathbf{a}\rangle}{2\sigma^2})}{\sum_{\mathbf{a}:b_i=1} \exp(\frac{\langle r,\mathbf{a}\rangle}{2\sigma^2})}\right] \tag{2.5}$$

Each sum in (2.5) can be approximated by its largest term [4]. The largest term in the numerator is the nearest constellation point to r that has a 0 in bit position i, and the largest term in the denominator is the nearest constellation point that has a 1 in bit position i. This approximation is expressed in (2.6), where $\mathbf{a}^u_{\min} = a(\text{argmin}_{\mathbf{a}:b_i=u}\{||r-\mathbf{a}||^2\})$.

$$\lambda_i \approx \frac{1}{2\sigma^2}(2\,\text{Re}[\langle r, \mathbf{a}^0_{\min} - \mathbf{a}^1_{\min}\rangle] + ||\mathbf{a}^0_{\min}||^2 - ||\mathbf{a}^1_{\min}||^2) \tag{2.6}$$

It can be seen in (2.6) that the confidence of a particular bit decision increases as the distance between the nearest 0 and nearest 1 grows.

2.3.2 Decoding Algorithm

LDPC codes employ a sparse $N \times M$ parity check matrix H. The optimal decoding algorithm, described in [5], iteratively updates the nodes of the Tanner graph representing H. The algorithm's main source of complexity is the check node update shown in (2.7), where $N_{m,n}$ denotes the bits from the mth check, excluding bit n.

$$\eta^{[l]}_{m,n} = -2\tanh^{-1}\left(\prod_{i\in N_{m,n}} \tanh\left(-\frac{\lambda_i^{[l-1]} - \eta^{[l-1]}_{m,i}}{2}\right)\right) \tag{2.7}$$

In order to reduce the algorithm's complexity, the scaled-min algorithm can be used. This approach replaces (2.7) with the check node update shown in (2.8), where β is a constant.

$$\eta^{[l]}_{m,n} = -\beta\left(\prod_{i\in N_{m,n}} \text{sign}(-\lambda_i^{[l-1]} + \eta^{[l-1]}_{m,i}) \times \min_{i\in N_{m,n}}\{|-\lambda_i^{[l-1]} + \eta^{[l-1]}_{m,i}|\}\right) \tag{2.8}$$

The scaled-min algorithm was used for this system. The coding gain loss for the scaled-min algorithm in this case is 0.1 dB [6].

2.4 Constellation

Bits were mapped to 16-APSK symbols according to the specifications in [7]. The constellation consists of two concentric circles with 12 equally spaced points on the outer ring and 4 equally spaced points on the inner ring. The ratio of the outer radius to the inner radius is denoted γ and is determined in [7] by the LDPC code rate. For this system, in which the LDPC code rate was $4/5$, $\gamma = 2.75$. Reasons for choosing this particular LDPC code rate will be discussed later.

The 16-APSK constellation is shown in Figure 2.4. The leftmost bit is considered the most significant bit.

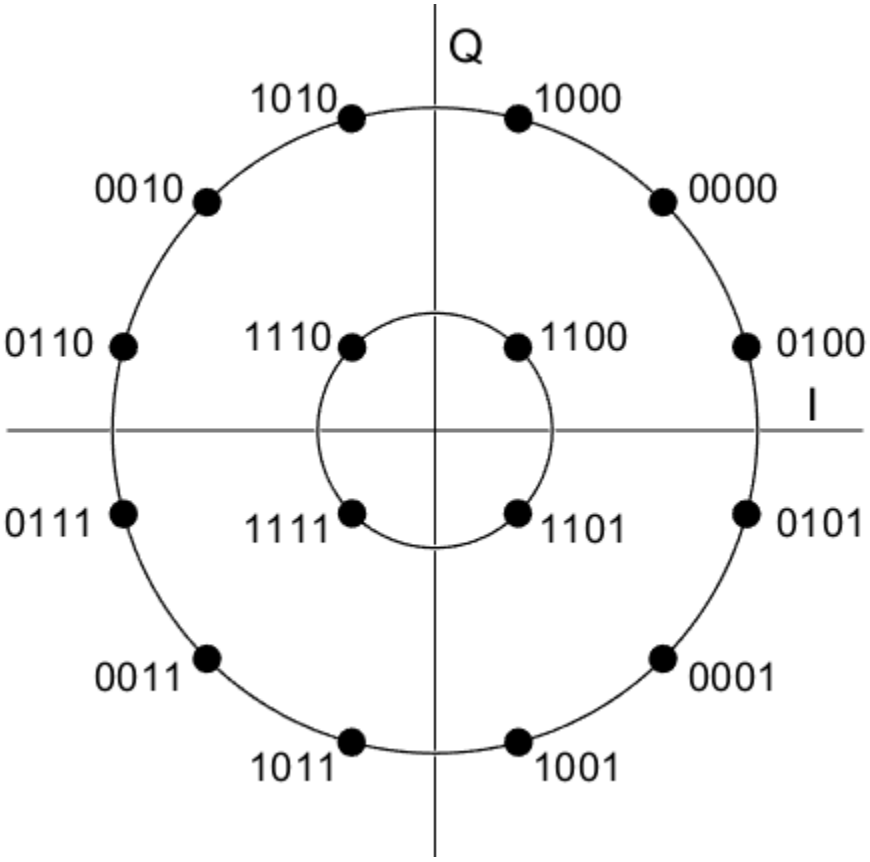

Figure 2.4: 16-APSK Constellation

Chapter 3

Theory of Operation

This chapter explains the theory of operation of the phase-locked loop (PLL)-based synchronization system as presented in [2]. The discussion begins after the output of the receiver's matched filter has been downsampled; these downsampled matched filter outputs are the inputs to the synchronization system. The systems are first presented independently of each other, i.e. perfect timing synchronization is assumed during the phase synchronization system discussion and vice versa. Under this assumption, it is valid to use the symbol time index T_s for samples in the phase synchronization system.

3.1 Carrier Phase Synchronization

3.1.1 Presentation of Problem

A carrier phase offset means that the oscillator at the receiver is not aligned in phase and/or frequency with the oscillator at the transmitter. This results in the matched filter output being out of phase with its nearest constellation point (i.e. the decision) as shown in Figure 3.1, even in a noiseless environment. Figure 3.2 shows simulation results for a noiseless 16-APSK system where an offset in phase and frequency was introduced between the transmit and receive oscillators. The black X's mark the 16-APSK constellation points. The effect of the frequency offset can be seen in that the matched filter outputs appear to be spinning rather than forming a constellation.

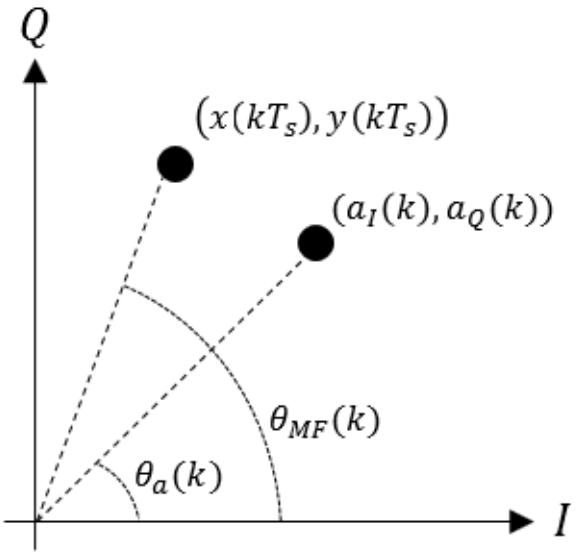

Figure 3.1: Graphical Interpretation of Phase Error

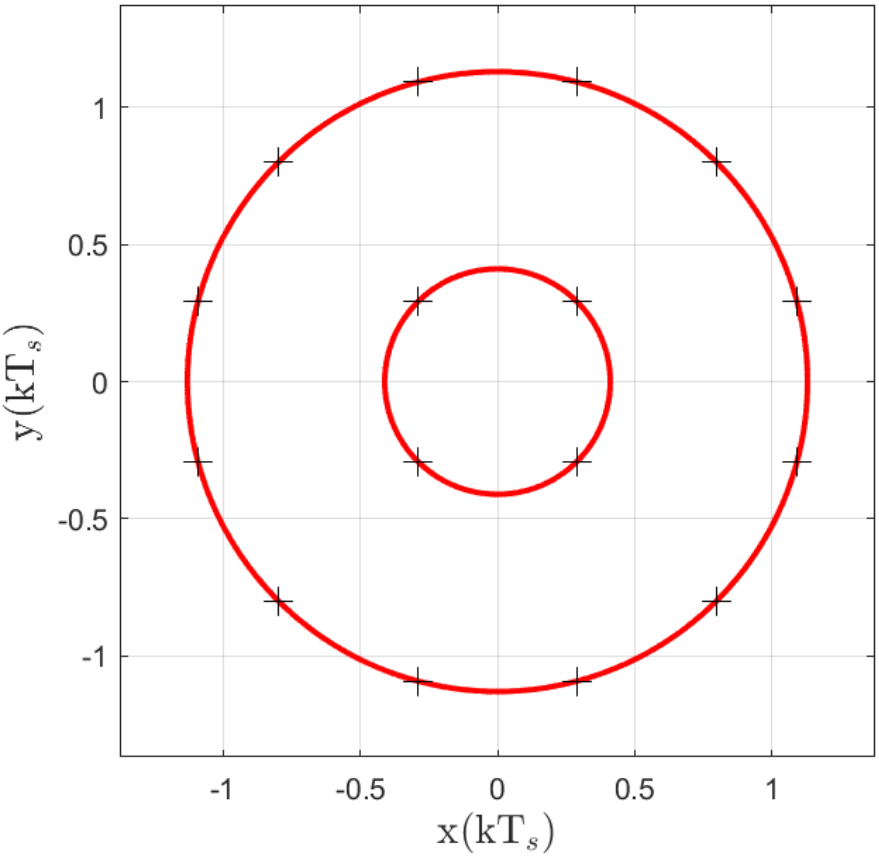

Figure 3.2: Demodulation with Phase and Frequency Offset, No Synchronization System

This spinning effect can be explained mathematically from the complex multiplication that takes place at the transmit and receive mixers. The oscillator at the transmitter's mixer is defined as $e^{-j\Omega_{tx}n}$. If the receiver mixer's oscillator is $e^{-j(\Omega_{rx}n+\phi)}$, where $\Omega_{tx} \neq \Omega_{rx}$ and $\phi \neq 0$, then the received signal is being rotated at a different rate (and with a different starting point) than the transmitted signal was. The job of the phase synchronization system is to track and compensate for this offset.

Therefore, the phase synchronization system can be seen as a rotation of matched filter outputs in order to properly align them into a constellation. This rotation is achieved using the counter-clockwise rotation matrix shown in (3.1), where $\hat{\theta}$ is the estimated carrier phase offset.

$$\begin{bmatrix} x'(kT_s) \\ y'(kT_s) \end{bmatrix} = \begin{bmatrix} \cos(\hat{\theta}(k)) & -\sin(\hat{\theta}(k)) \\ \sin(\hat{\theta}(k)) & \cos(\hat{\theta}(k)) \end{bmatrix} \begin{bmatrix} x(kT_s) \\ y(kT_s) \end{bmatrix} \tag{3.1}$$

3.1.2 PLL

In order to estimate the carrier phase offset, the phase synchronization system used in this work employs the phase-locked loop (PLL). In general, a PLL attempts to drive the phase difference between its input and reference signal to 0. The fundamental continuous-time PLL achieves this with a phase detector, loop filter, and voltage controlled oscillator (VCO). The phase detector output is a function of the phase difference between the input and reference. The phase detector output is filtered by the loop filter, which is designed to have a nonzero gain for a constant phase offset and an infinite DC gain for a constant frequency offset. The VCO outputs a signal whose frequency is controlled by the filter output. The VCO output is the input to the phase detector, completing the loop. The discrete-time PLL used in this work is based on the continuous-time PLL and consists of three components: a phase error detector (PED), a loop filter, and a direct digital synthesizer (DDS).

The inputs to the PED are the rotated matched filter output $r'(kT_s) = x'(kT_s) + jy'(kT_s)$ (which has

been rotated by the previous PLL phase estimate) and the decision $\hat{a}(k)$ made on $r'(kT_s)$. If $\theta'(k)$ is the phase of $r'(kT_s)$, then a simple PED could be formulated as shown in (3.2) using the depiction of phase error in Figure 3.1. Here, the point $a(k)$ in Figure 3.1 was taken as the decision $\hat{a}(k)$.

$$\begin{aligned} e(k) &= \theta'(k) - \theta_{\hat{a}}(k) \\ &= \tan^{-1}\left(\frac{y'(kT_s)}{x'(kT_s)}\right) - \tan^{-1}\left(\frac{\hat{a}_Q(k)}{\hat{a}_I(k)}\right) \end{aligned} \tag{3.2}$$

The PED in (3.2) requires two divisions and two inverse tangent operations. A more sophisticated PED which eliminates these expensive computations, the maximum-likelihood (ML) PED, is derived in [2] and used in this work. The ML PED minimizes the value of $e(k)$ with respect to the conditional probability of the received vector given its phase. The ML PED is shown in (3.3) and Figure 3.3.

$$e(k) = y'(kT_s)\hat{a}_I(k) - x'(kT_s)\hat{a}_Q(k) \tag{3.3}$$

The PED output is fed to the loop filter. In order to achieve the loop filter performance specified earlier, i.e. nonzero gain for a constant phase offset and infinite DC gain for a constant frequency offset, the proportional-plus-integrator (PPI) filter is used in this work. The PPI filter has the transfer function $H(s) = k_1 + \frac{k2}{s}$, or $H(z) = K_1 + K_2\frac{1}{1-z^{-1}}$, where K_1 and K_2 are gain constants to be discussed later. Using the PPI filter results in a second-order PLL. A block diagram of this filter can be seen in Figure 3.3.

The loop filter output is the input to the DDS. The phase of the DDS output is the scaled accumulation of the loop filter outputs $v(k)$, i.e. $\hat{\theta}(k+1) = K_0\sum v(k)$, where K_0 is a gain constant which determines the sensitivity of the DDS. The value $\hat{\theta}(k+1)$ is the phase correction estimate for the next matched filter output.

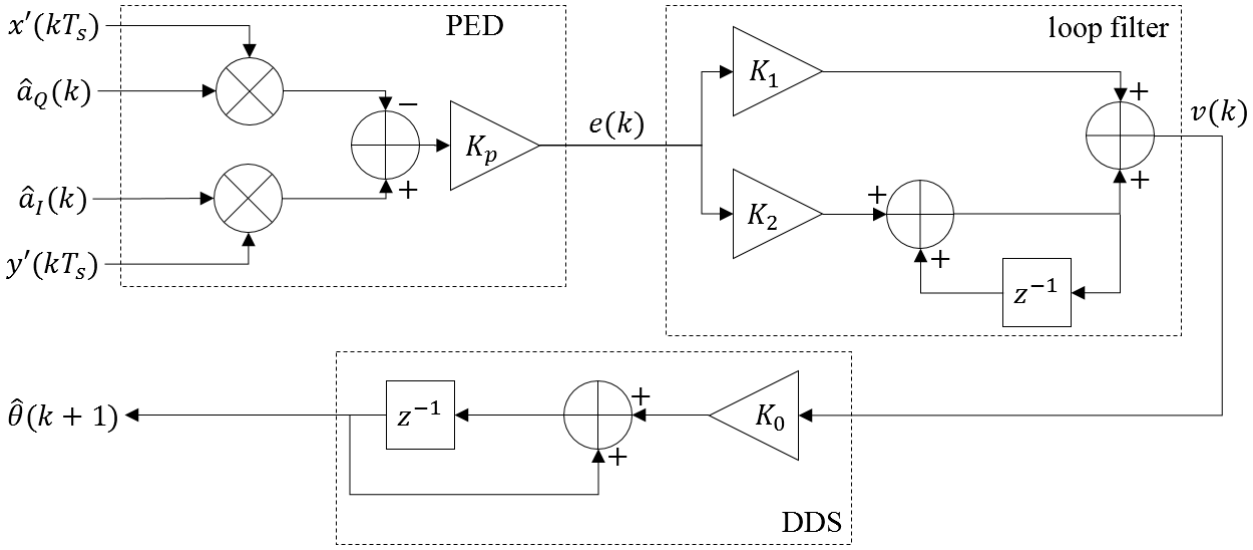

Figure 3.3: Second-Order PLL with PPI Filter for Carrier Phase Synchronization

The gain constants K_p, K_0, K_1, and K_2 are derived from the PLL loop transfer function and determine the performance of the PLL. K_p is the slope of the PED S-curve at $\theta_e = 0$ (S-curves will be discussed in Section 3.1.4). K_0 determines the DDS sensitivity as previously mentioned. The values of K_1 and K_2 can be well approximated by (3.4) and (3.5), where ζ is the loop damping factor and B_nT is the loop bandwidth [2].

$$K_pK_0K_1 \approx \frac{4\zeta}{\zeta + \frac{1}{4\zeta}} B_nT \tag{3.4}$$

$$K_pK_0K_2 \approx \frac{4}{(\zeta + \frac{1}{4\zeta})^2} (B_nT)^2 \tag{3.5}$$

3.1.3 Phase Synchronization System

Figure 3.4 shows a block diagram of the phase synchronization system as a whole. The rotation block carries out (3.1), and the PLL block is the block diagram of Figure 3.3.

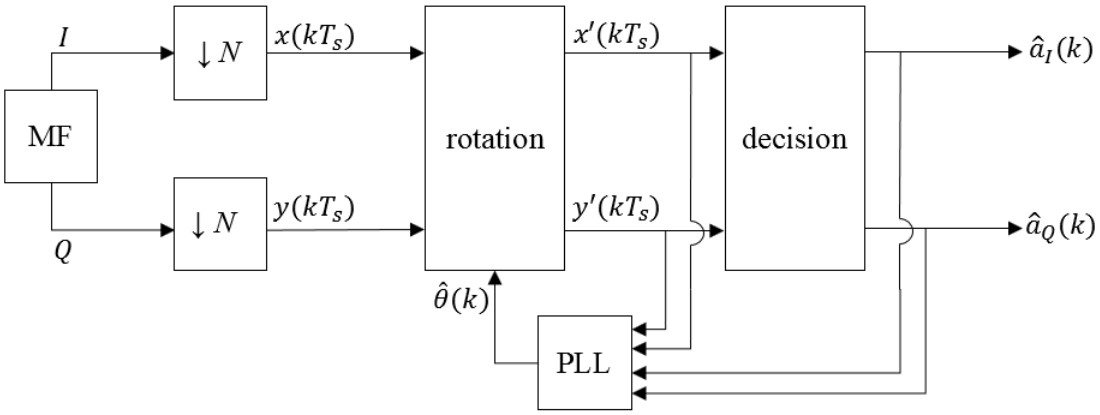

Figure 3.4: Phase Synchronization System

3.1.4 Phase Ambiguity Resolution

It is inherently possible for the PLL to lock onto a rotated version of the constellation as it is acquiring the phase of the received signal. If the initial phase offset at the receiver is great enough to move a sample into a different decision region, then the PLL will compensate with respect to the decision rather than the true transmitted symbol. This will result in a set of decisions that matches the shape of the desired constellation, but each symbol will in fact include a constant phase offset despite the PED output approaching zero. This property is known as the PLL's phase ambiguity.

The set of possible phase ambiguities depends on the modulation scheme. Potential phase ambiguities can be predicted by observing a constellation's symmetry; however, a more mathematical approach is to generate an S-curve for a given system. An S-curve, denoted $g(\theta_e)$, plots the PED output $e(k)$ versus the phase error θ_e.

To obtain $g(\theta_e)$, the expression for $e(k)$ shown in (3.3) is rewritten in terms of θ_e by substituting $x'(kT_s)$ and $y'(kT_s)$ from (3.1). The resulting equation is shown in (3.6) for a given decision.

$$g(\theta_e) = (x(kT_s)\sin(\theta_e) + y(kT_s)\cos(\theta_e))\hat{a}_I(k) - (x(kT_s)\cos(\theta_e) - y(kT_s)\sin(\theta_e))\hat{a}_Q(kT_s) \quad (3.6)$$

The average S-curve taken over all possible symbols is denoted $\overline{g}(\theta_e)$. As stated in [2], the lock points of a PLL occur at the points where $\overline{g}(\theta_e)$ crosses the x-axis with a positive slope. In general,

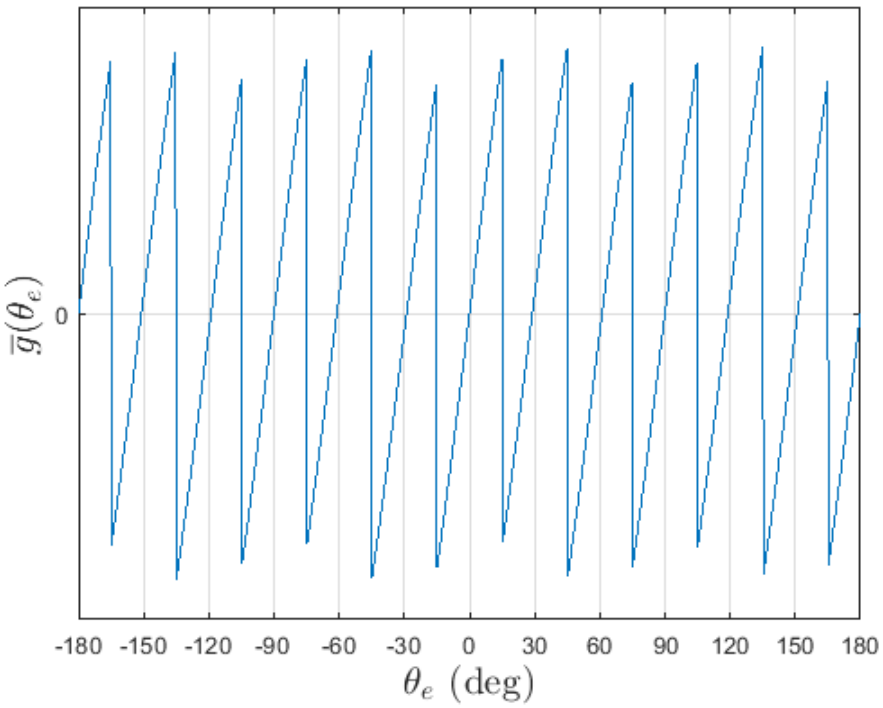

Figure 3.5: Average S-Curve for 16-APSK

if $\overline{g}(\theta_e)$ has L positive zero crossings, then the PLL is said to have a $2\pi/L$ phase ambiguity. For example, in a QPSK system, the average S-curve has positive zero crossings at $\{-\pi/2, 0, \pi/2, \pi\}$, resulting in a $\pi/2$ phase ambiguity. This is expected because the QPSK constellation has $\pi/2$ rotational symmetry.

The average S-curve for this work's 16-APSK constellation is shown in Figure 3.5. The 12 positive zero crossings indicate a $\pi/6$ phase ambiguity for the system. Although the 16-APSK constellation has true $\pi/2$ rotational symmetry, the $\pi/6$ phase ambiguity is due to the symmetry of the outer ring of the constellation. Because a majority of the symbols fall on the outer ring, and because these points are much larger in magnitude than the inner points, $g(\theta_e)$ for the inner ring symbols has a relatively small effect on $\overline{g}(\theta_e)$.

In order to allow proper decoding, the phase ambiguity must be resolved. For this work, phase ambiguity resolution was achieved by adding an attached synchronization marker (ASM) to the

beginning of each transmitted LDPC frame as specified in [8]. Details of the ASM will be discussed later. For phase ambiguity resolution, the ASM is used to provide a known phase reference for the demodulated data. The ASM is known at the receiver; therefore, if a rotated version of the ASM is detected, then an additional correction can be added to the PLL to compensate for the incorrect phase lock.

The ASM is optimally detected using a correlation detector [9]. The correlation detector marks the location of the ASM when the correlation between the ASM and the demodulated symbols exceeds a given threshold. However, to avoid the computationally expensive correlation calculation, an alternative approach is to consider bit sequences rather than data vectors. Using the phase ambiguity found from the average S-curve, rotated versions of the ASM can be computed for each possible PLL lock point. Then, these symbols are run through a decision block to find the bit sequence for each version of the ASM. Denote the length of the ASM as N_{ASM} bits; as bit decisions are made at the receiver, the most recent N_{ASM} bits are compared to each length-N_{ASM} ASM bit sequence. If the number of differences is less than a given threshold (analogous to the threshold of the correlation detector) for any rotated ASM, then the phase offset of that ASM is considered the positive zero crossing from $\overline{g}(\theta_e)$ at which the PLL locked. This phase offset can then be corrected in the next iteration of the PLL.

3.2 Symbol Timing Synchronization

3.2.1 Presentation of Problem

A symbol timing error is introduced when the received signal is sampled by the ADC in between ideal sampling instants. For a discrete-time signal upsampled by N, every Nth point corresponds to a data symbol. After this signal is converted to the continuous-time domain for transmission, the received analog signal then needs to be sampled at the correct instant in time in order to perfectly recover the transmitted data symbol. This ideal time instant for the kth symbol will be denoted kT_0. At time kT_0, the signal has maximum amplitude, and consequently the highest average SNR,

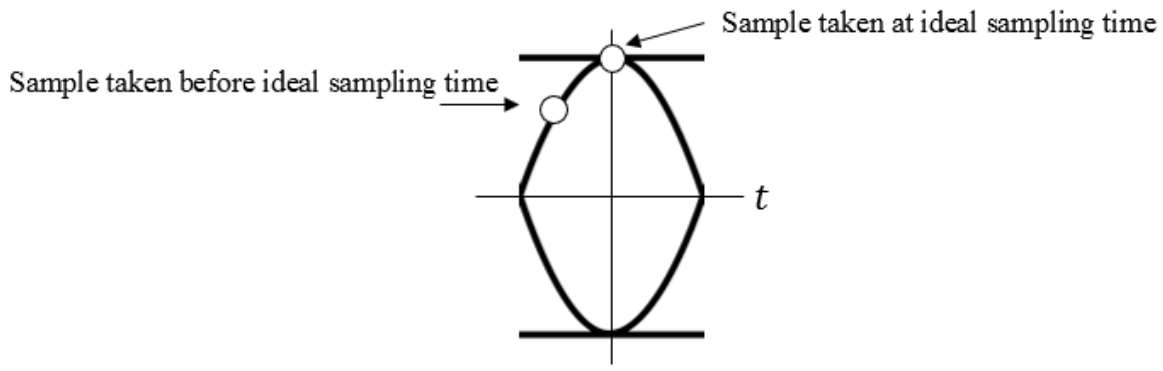

Figure 3.6: Graphical Representation of Timing Error

for that symbol time. However, even in the absence of noise, sampling the received signal at the correct times is critical because the sample may otherwise lie in the incorrect decision region, resulting in communication errors. This concept is best depicted by a signal's eye diagram as in Figure 3.6. The ideal sampling time is at the eye diagram's maximum opening. A symbol timing error occurs when the signal is sampled before of after the maximum eye diagram opening.

Sampling at exactly time kT_0 for every symbol is not practical in real systems. An ADC clock is not a perfect timing source, so there is inevitably a small timing error in the sampling rate of the ADC. The main reason for imperfect symbol timing is that the receiver ADC most likely did not start sampling at time 0 of the received signal, so even if the sample rate was exactly $1/T_0$, the samples would not be at ideal sampling times.

Figure 3.7 shows simulation results for a noiseless 16-APSK system where a symbol timing offset was introduced at the receiver. The timing offset was simulated by sampling the received signal at the perfect rate $1/T_0$, but at time $(k - \frac{1}{16})T_0$. Even without added noise, the demodulated signal does not form a constellation because the samples do not correspond to 16-APSK symbols.

The goal of a symbol timing synchronization system is to ensure that during each symbol time, there is a sample aligned with the eye diagram's maximum opening. However, the system does not achieve synchronization by physically changing the sampling times of the ADC. In general, the ADC sampling rate is fixed, and the timing synchronization system does not control the ADC.

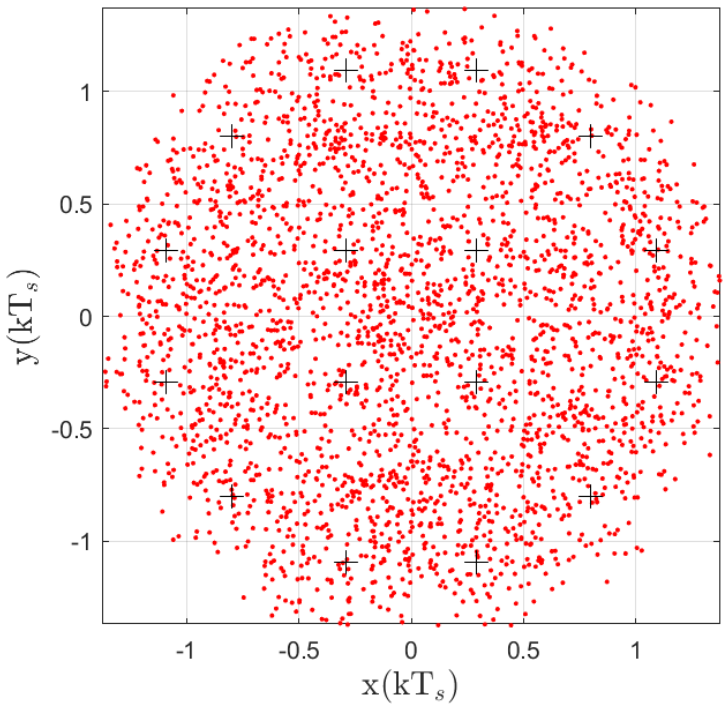

Figure 3.7: Demodulation with Symbol Timing Offset, No Synchronization System

Therefore, the synchronization system is tasked with extracting the original timing information from the ADC output.

3.2.2 Interpolator

One approach to correcting a symbol timing offset is to sample the received signal at a rate higher than the symbol rate and interpolate the samples. This estimates what the value of a sample would have been if it were taken at the ideal sampling instant. This can again be seen graphically with an eye diagram as in Figure 3.6: If multiple samples are taken on both sides of the maximum eye opening in the same symbol time, then the samples can be interpolated to find the value at the maximum eye opening.

Several important variables will be introduced to allow explanation of the interpolation process. If kT_0 is the ideal sampling instant for the kth symbol, then the time $t(k)$ is the time at which the sample $r(t(k))$ for the kth symbol was taken. The time instant $t(k)$ is different than kT_0 by a fraction

of a symbol time denoted $\mu(k)$. Therefore, the estimation of $\mu(k)$ is the main task of the timing synchronization system. This will be discussed in Section 3.2.3.

Numerous interpolation methods exist. The piecewise quadratic interpolator was used in this work due to its relatively low computational complexity. The piecewise quadratic interpolator using four points for interpolation can be described as a filter as shown in (3.7) with filter coefficients h given in (3.8). Although a quadratic interpolator only requires three points, an even number of points is needed for the filter to be symmetric about $\mu(k) = 1/2$ and have linear phase.

$$r((t(k)+\mu(k))T) = \sum_{i=-2}^{1} h(i)r(t(k-i)T) \tag{3.7}$$

$$\begin{aligned} h(-2) &= \alpha\mu(k)^2 - \alpha\mu(k) \\ h(-1) &= -\alpha\mu(k)^2 + (1+\alpha)\mu(k) \\ h(0) &= -\alpha\mu(k)^2 - (1-\alpha)\mu(k) + 1 \\ h(1) &= \alpha\mu(k)^2 - \alpha\mu(k) \end{aligned} \tag{3.8}$$

The constant α in (3.8) is a free parameter resulting from extending the interpolator input to four points. Using $\alpha = 1/2$ offers near-optimal performance and simple implementation [10]. Still, the expense of (3.7) can be further reduced by rewriting (3.7) as a Farrow-structured filter [11]. The coefficients in (3.8) can be written in terms of $\mu(k)$, resulting in the interpolator in (3.9) with constant coefficients b given in Table 3.1.

$$r((t(k)+\mu(k))T) = \sum_{p=0}^{2} \mu(k)^p \sum_{i=-2}^{1} b_p(i)r(t(k-i)T) \tag{3.9}$$

A block diagram of the Farrow piecewise quadratic interpolator is shown in Figure 3.8.

Table 3.1: Constant Filter Coefficients for Farrow Piecewise Quadratic Interpolator

i	$b_2(i)$	$b_1(i)$	$b_0(i)$
-2	α	$-\alpha$	0
-1	$-\alpha$	$\alpha+1$	0
0	$-\alpha$	$\alpha-1$	1
1	α	$-\alpha$	0

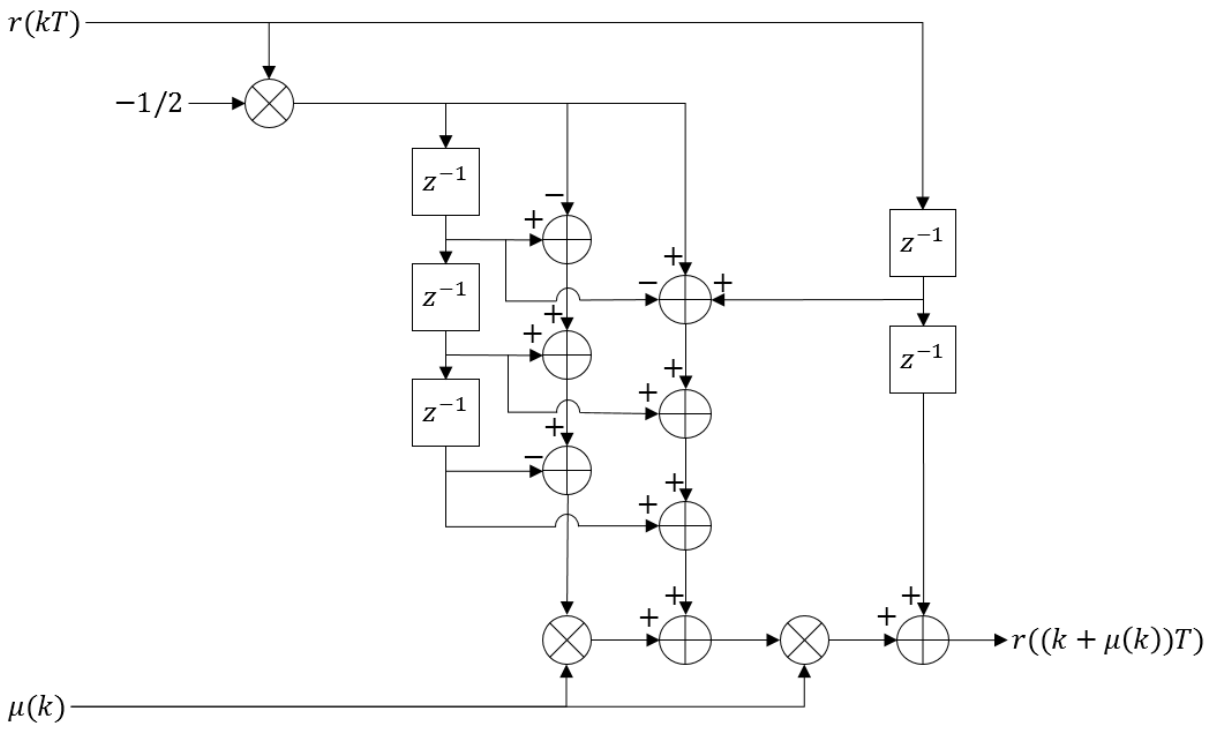

Figure 3.8: Farrow Piecewise Quadratic Interpolator

3.2.3 Interpolation Control

The interpolator is controlled by the fractional interval $\mu(k)$. As described earlier, $\mu(k)$ estimates the fraction of a symbol time between the sampled time instant $t(k)$ and the ideal sampling time instant kT_0. The method used in this work for estimating $\mu(k)$ is based on the PLL and is similar to the estimation of $\hat{\theta}$ in the phase synchronization system. There are two main differences between the symbol timing PLL and carrier phase PLL: The timing error detector (TED) in place of the PED and the modulo-1 decrementing counter in place of the DDS. Because the interpolator operates at a rate faster than the symbol rate, the rate of the timing PLL must also be faster than the symbol rate.

There are a number of TEDs described in [2]. The zero-crossing timing error detector (ZCTED) was used in this work. The ZCTED operates at 2 samples per symbol and outputs a timing error estimate once per symbol. It is based on the idea that a sample $r(t(k))$ was sampled at the eye diagram's maximum eye opening if the previous and next sample, i.e. $r(t(k-1/2))$ and $r(t(k+1/2))$, are both 0. The ZCTED error signal is shown in (3.10) in terms of the rotated matched filter output $x'(kT_s)+jy'(kT_s)$. The timing in (3.10) is expressed in terms of the symbol index k and the symbol time T_s for timing offset τ to emphasize the 2 samples per symbol operating rate. The TED output approaches 0 as the sample at time $(k-1/2)T_s$, i.e. the sample between symbol times, approaches 0. The difference $\hat{a}(k-1)-\hat{a}(k)$ stems from an approximation of the derivative of the eye diagram used in the ML TED.

$$e(k) = x((k-1/2)T_s+\tau)[\hat{a}_I(k-1)-\hat{a}_I(k)]+y((k-1/2)T_s+\tau)[\hat{a}_Q(k-1)-\hat{a}_Q(k)] \qquad (3.10)$$

The same PPI loop filter can be used in the symbol timing PLL as was described for the carrier phase PLL. The only modifications are the gain constants K_0, which must be negative due to the decrementing counter, and K_p. In the case of the TED, the value of K_p is a function of both the signal energy and the pulse shape. A plot of K_p as a function of the root rasied cosine pulse's excess bandwidth is given in [2].

The loop filter output is fed to the modulo-1 decrementing counter, which is analogous to the modulo-2π DDS in the phase recovery PLL. The modulo-1 decrementing counter is a significant component of the entire synchronization system because it is raises a flag ("strobe") at the symbol rate to clock the system outputs. In other words, this strobe marks each estimated ideal sampling time. The strobe goes high whenever the counter underflows. Therefore, the counter is designed to underflow every D samples, where D is the rate of the synchronization system in samples per symbol (e.g. $D = 2$ if the ZCTED is used). The counter decrements by an average of $1/D$ each iteration. The amount by which it decrements is controlled by the loop filter output. The counter operation is shown in (3.11), where $\eta(k+1)$ is the counter output and $v(k)$ is the filter output.

$$\eta(k+1) = \left(\eta(k) - \left(v(k) + \frac{1}{D}\right)\right) \mod 1 \tag{3.11}$$

When the counter underflows, the value of μ is updated as shown in (3.12).

$$\mu(k+1) = \frac{\eta(k)}{v(k) + \frac{1}{D}} \tag{3.12}$$

A block diagram of the timing synchronization system PLL is shown in Figure 3.9. The stars indicate the blocks which are enabled by the counter's strobe signal. The TED output is 0 when the strobe is low. The value of μ is only updated when the strobe is high; when the strobe is low, $\mu(k+1) = \mu(k)$.

3.2.4 Timing Synchronization System

Figure 3.10 shows a block diagram of the timing synchronization system as a whole. The decision block needs the strobe so that decisions are only made on incoming samples that correspond to valid symbols. Downsampling by $N/2$ results in $D = 2$ samples per symbol, the rate of the ZCTED.

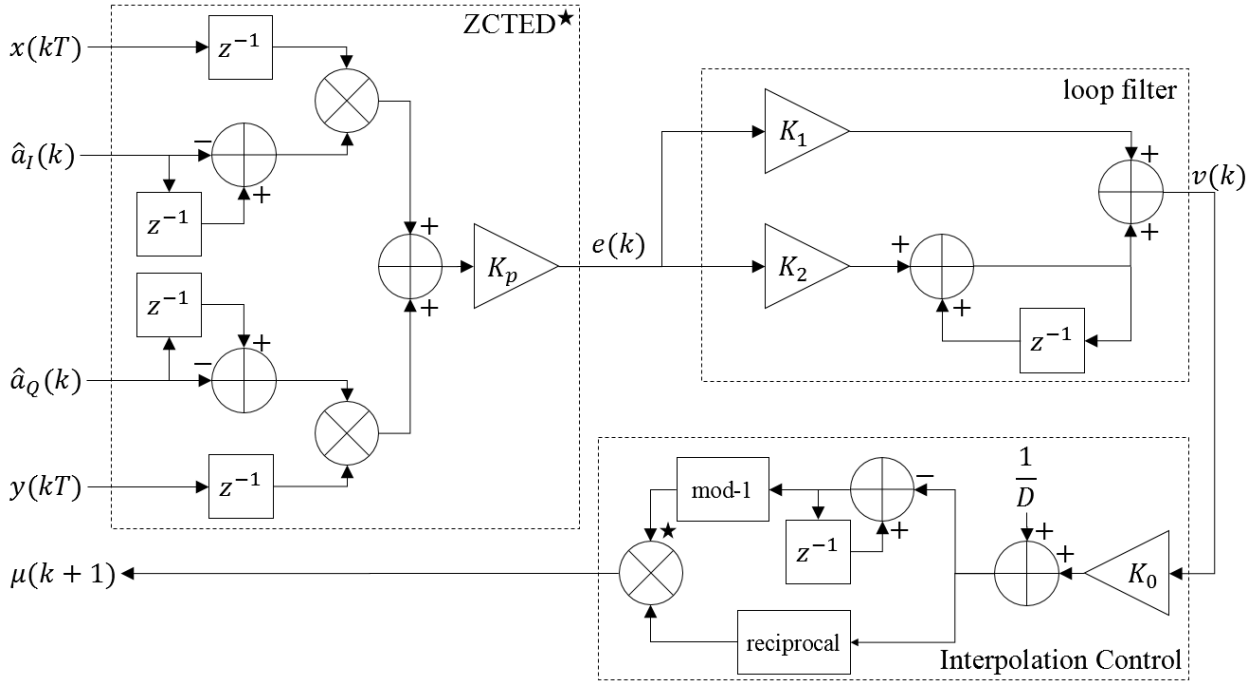

Figure 3.9: Second-Order PLL with PPI Filter for Timing Synchronization

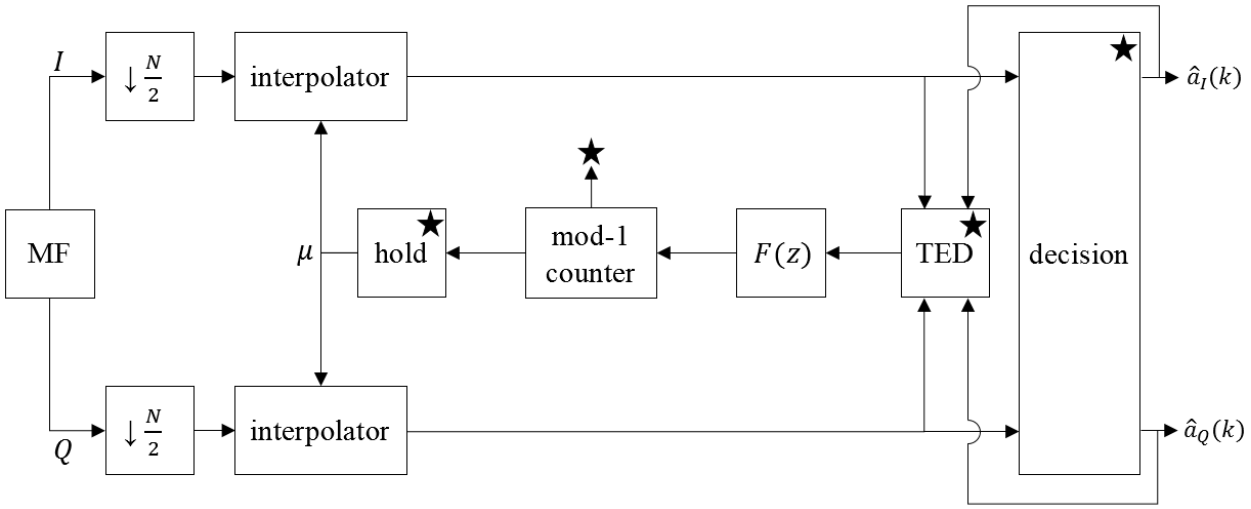

Figure 3.10: Timing Synchronization System

3.3 Combined Carrier Phase and Symbol Timing Synchronization

The complete synchronization system combines the carrier phase recovery system of Figure 3.4 and the symbol timing synchronization system of Figure 3.10. The phase recovery system was previously discussed in terms of a system with perfect symbol timing operating at 1 sample per symbol. However, when combined with the timing synchronization system, the components of the phase recovery system must now operate at a higher sample rate. Consequently, the strobe signal discussed for the timing synchronization system is now also applied to the components of the phase synchronization system.

The combined carrier phase and symbol timing synchronization system is shown in Figure 3.11.

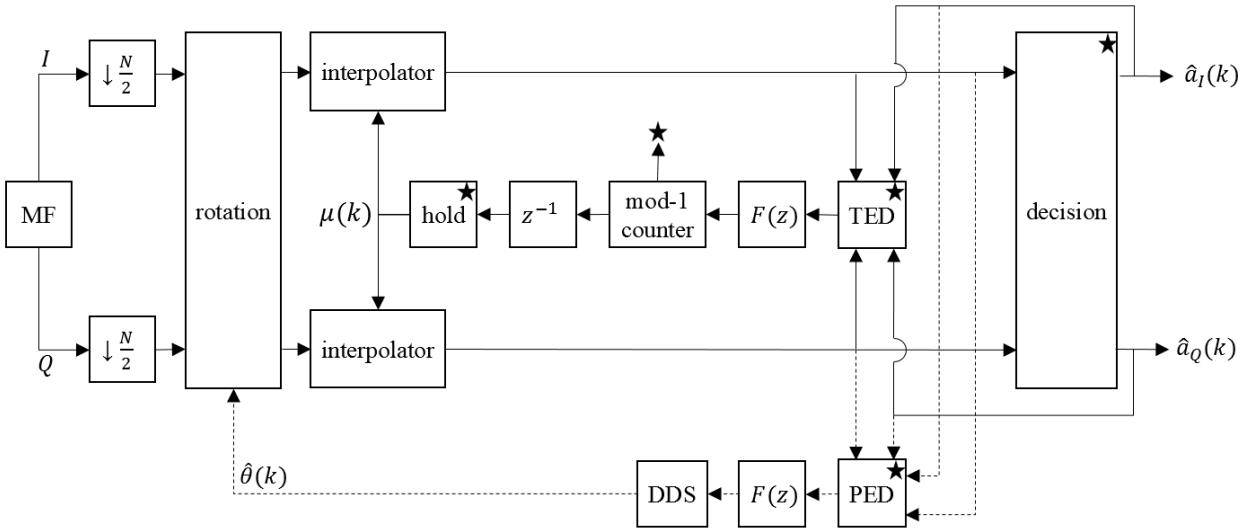

Figure 3.11: Carrier Phase and Symbol Timing Synchronization System

Chapter 4

Hardware Implementation

This chapter details how each block of the receiver was implemented on the FPGA. The first section refers to the synchronization system of Figure 3.11. The second section discusses the implementation of other necessary components of the receiver. Finally, the FPGA clock and reset signals are discussed. Unless otherwise noted, calculations were performed using 16-bit fixed point numbers.

4.1 Synchronization System Implementation

4.1.1 Counterclockwise Rotation with CORDIC

The rotation of the filter outputs by the estimated phase offset was performed using the coordinate rotational digital computer (CORDIC) algorithm. As described in [12], the CORDIC algorithm for vector rotation performs an iterative approximation of the rotation matrix in (3.1). Using the trigonometric identities $\cos(\theta) = \frac{1}{\sqrt{1+\tan^2(\theta)}}$ and $\sin(\theta) = \frac{\tan(\theta)}{\sqrt{1+\tan^2(\theta)}}$, the rotation expressed in (3.1) can be rewritten as shown in (4.1) for the i_{th} CORDIC iteration.

$$\begin{bmatrix} x_i'(kT_s) \\ y_i'(kT_s) \end{bmatrix} = \frac{1}{\sqrt{1+\tan^2(\theta_i)}} \begin{bmatrix} 1 & -\tan(\theta_i) \\ \tan(\theta_i) & 1 \end{bmatrix} \begin{bmatrix} x_{i-1}(kT_s) \\ y_{i-1}(kT_s) \end{bmatrix} \tag{4.1}$$

By constraining θ_i such that $\tan(\theta_i) = \pm 2^{-i}$, (4.1) can be rewritten as shown in (4.2), where $\alpha_i =$

± 1.

$$\begin{bmatrix} x_i'(kT_s) \\ y_i'(kT_s) \end{bmatrix} = \frac{1}{\sqrt{1+2^{-2i}}} \begin{bmatrix} 1 & -\alpha_i 2^{-i} \\ \alpha_i 2^{-i} & 1 \end{bmatrix} \begin{bmatrix} x_{i-1}(kT_s) \\ y_{i-1}(kT_s) \end{bmatrix} \tag{4.2}$$

This constraint on θ_i results in the relationship shown in (4.3).

$$\theta_{i+1} = \theta_i + \alpha_i \arctan(2^{-i}) \tag{4.3}$$

The only variable in (4.3) is the sign of α_i. This is determined by the error $\tilde{\theta}$ between the desired rotation angle ($\hat{\theta}$ in this case) and the accumulation of the previous rotation angles as shown in (4.4). The goal of the algorithm is to force $\tilde{\theta}$ to converge to 0. Therefore, if $\theta_i > 0$, then α_{i+1} is set to -1, and vice versa.

$$\tilde{\theta}_i = \hat{\theta} - \theta_i \tag{4.4}$$

The advantage of this algorithm can be seen in the reduced computational complexity between (3.1) and (4.2): While (3.1) requires four multiplications and two additions, (4.2) replaces the multiplications with divisions by powers of two, which can be easily implemented with bit-shifts. Furthermore, a smaller lookup table is required for the CORDIC algorithm because for n iterations, only n angle values need to be stored instead of a complete sine lookup table.

The accuracy of the CORDIC algorithm depends on n. An example of the tradeoff between accuracy and latency is quantified in Figure 4.1 for a rotation of the real-valued vector $1+0j$ by $\pi/2$ radians. This convergence rate was consistent for several test rotations. Based on this performance and the maximum allowable latency of the CORDIC block, the number of iterations used on the FPGA was chosen as $n = 11$.

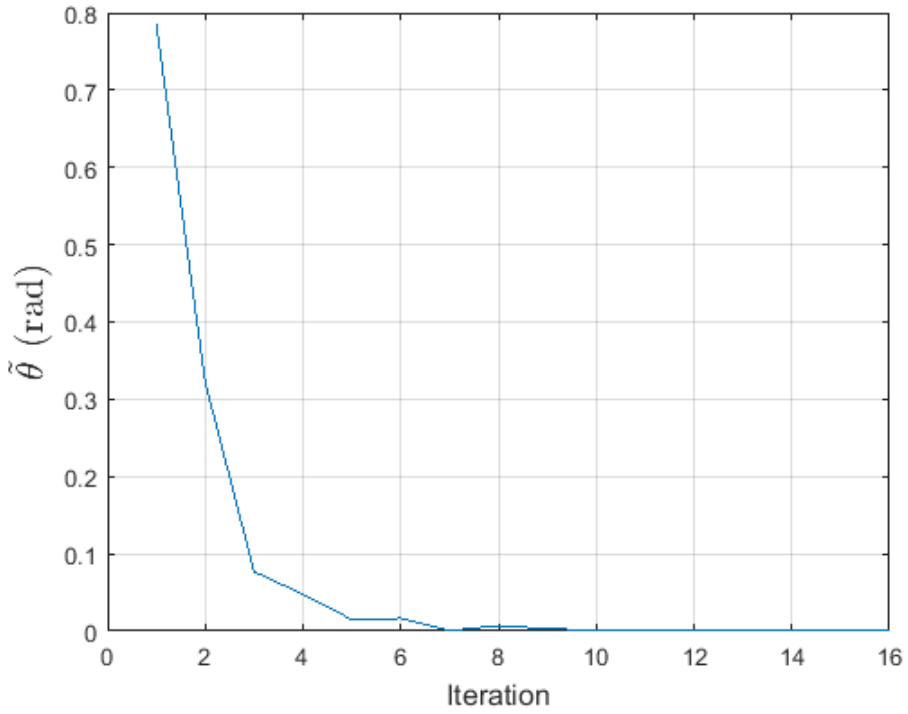

Figure 4.1: CORDIC Rotation Angle Error Convergence

The CORDIC scale factor Z mentioned in Section 4.2.2 is intended to compensate for the $\frac{1}{\sqrt{1+2^{-2i}}}$ term in (4.2). As previously stated, latency and resources can be reduced by ignoring this term in the algorithm to eliminate a multiplication. Ignoring this term means that the CORDIC output will be scaled by the value Z, which is a function of n as shown in (4.5). After choosing n, the value of the inverse of Z can be computed for scaling of the CORDIC inputs to obtain rotated vectors of the expected average amplitude.

$$Z = \prod_{i=1}^{n} \sqrt{1+2^{-2i}} \tag{4.5}$$

4.1.2 Farrow Piecewise Quadratic Interpolator

The interpolator shown in Figure 3.8 introduces the largest latency of any of the timing synchronization subsystems due to the large number of multiplications. Therefore, minimizing latency

was the main focus of the interpolator implementation. This was achieved by performing any calculations in advance that only depended on past inputs before the next loop began.

4.1.3 Decisions

The interpolated outputs are used to make bit decisions. The maximum-likelihood decision is made by finding the constellation point with the minimum squared Euclidean distance from the received symbol on the complex plane. This is shown in (4.6), where $\hat{a}$ is the decision, $\mathbb{A}$ is the set of complex constellation points, and r is the complex interpolator output.

$$\hat{a} = \underset{a \in \mathbb{A}}{\operatorname{argmin}} \{ \|r - a\|^2 \} \tag{4.6}$$

The computation in (4.6) can be simplified to remove complex multiplications by expanding the squared Euclidean distance term to $\|r-a\|^2 = \|r\|^2 - 2\langle r, a \rangle + \|a\|^2$. The $\|r\|^2$ term can be ignored in the argmin argument since it is a constant. Therefore, (4.6) can be rewritten as (4.7).

$$\hat{a} = \underset{a \in \mathbb{A}}{\operatorname{argmax}} \{ \langle r, a \rangle - \frac{1}{2}\|a\|^2 \} \tag{4.7}$$

The $\frac{1}{2}\|a\|^2$ terms can be stored as constants for each constellation point, so the squared Euclidean distance computation now only requires two multiplications and two additions for each constellation point.

In this case, FPGA resource utilization can be further reduced due to the bit mapping of the constellation from Figure 2.4. In the symmetrical 16-APSK constellation, the mapping of the two most significant bits is the same for points in different quadrants with equivalent I and Q magnitudes. Furthermore, the two least significant bits of each quadrant can be represented by the sign bits of the I and Q interpolator outputs. Consequently, for FPGA implementation, the absolute value of

the interpolator output is used as x in (4.7), and the set $\mathbb{A}$ can be restricted to the constellation points in the first quadrant. This decreases the number of multipliers and adders required, and it reduces latency by only finding the argmax of four values instead of 16.

4.1.4 Timing Error Detector

The ZCTED error signal in (3.10) was implemented on the FPGA with the main goal being low latency. By recognizing that the error signal depends only on the interpolator output from the previous run and the current decision, a length-16 lookup table of all possible error signals for the next iteration given the current interpolator output can be created. After a decision is made, the corresponding error signal for that decision is selected from the lookup table.

4.1.5 Phase Error Detector

The PED error signal from (3.3) was directly implemented on the FPGA. Latency was not a concern here because of the timing of the system as a whole: After the interpolator and decision blocks, there is a limited number of clock cycles until the next CCW rotation output is available for processing. The PLL requires a considerable number of clock cycles, and there would not be enough time for the PLL to output the phase correction estimate before the next sample was ready to receive the phase correction.

Therefore, a $\frac{1}{2}$ symbol time delay was built into the phase synchronization system here. After the PED computes the phase error, the PLL does not begin processing it until the next synchronization loop begins. This lag in phase correction was not found to impact system performance. This resulted in ample time for the PED to compute the phase error, so FPGA resources and performance were emphasized rather than minimizing latency.

4.1.6 PPI Filter

The second-order PPI filter seen in Figure 3.3 was implemented using 32-bit fixed point numbers for the loop gain constants. The extra precision was required here due to the small magnitude of the PLL gain constants. This came at the cost of additional DSP slices because the bit growth from the multiplication exceeded the size limit of a single slice. The parameters for both PLLs, calculated from (3.4) and (3.5), are shown in Table 4.1 and Table 4.2. The small loop bandwidth was selected for improved performance at lower SNR.

Table 4.1: Parameters for Phase Synchronization PLL

ζ	0.7071
B_nT	$1E-3$
K_0	1
K_p	1
K_1	$2.667E-3$
K_2	$3.556E-6$

Table 4.2: Parameters for Timing Synchronization PLL

ζ	0.7071
B_nT	$1E-3$
K_0	-1
K_p	2.68
K_1	$-9.950E-4$
K_2	$-1.327E-6$

The number of computations required for the filter of Figure 3.3 was reduced by recognizing that the output sum $K_1e(k)+K_2e(k)+\sum_{i=0}^{k-1}K_2e(i)$ can be rewritten as $(K_1+K_2)e(k)+\sum_{i=0}^{k-1}K_2e(i)$. In addition to reducing the number of required calculations, this also eliminated the need to store the value of K_1. Only the values of K_2 and (K_1+K_2) are needed.

4.1.7 DDS

Extra steps were required to make the DDS output shown in Figure 3.3 compatible with the CCW rotation block. The DDS output is the phase correction $\hat{\theta}$ for the CORDIC rotation algorithm;

however, the CORDIC algorithm was implemented to expect a value of $\hat{\theta}$ in the range $-\pi$ to π. The DDS output as shown in Figure 3.3 will continue to grow infinitely as its value accumulates with each iteration. To bound the value of $\hat{\theta}$, a block for wrapping the DDS output to $[-\pi, \pi]$ was added to the DDS.

Additionally, the DDS is where the phase ambiguity resolution is added to $\hat{\theta}$. When the ASM is detected, the corresponding phase correction is added to $\hat{\theta}(k-1)$ so the next $\hat{\theta}$ will include the adjustment.

4.1.8 Interpolator Update and Strobe

The latency of (3.11) was reduced by pre-computing $(\eta(k) - \frac{1}{2})$ before $v(k)$ was available. Before being wrapped by the modulo-1 operation, the sign bit from the decrementing counter of (3.11) sets the strobe high or low. The modulo-1 operation was implemented in hardware by simply setting the sign bit and integer bit of the decrementing counter low and keeping the fractional bits.

The sign bit of the decrementing counter also determined if the value of μ was updated. The μ update shown in (3.12) requires a division by $(v(k) + \frac{1}{2})$. However, in general, $v(k)$ is much smaller than $\frac{1}{2}$. Therefore, (3.12) can be well approximated by (4.8). This replaces the division with a bit-shift.

$$\mu(k) \approx 2\eta(k) \tag{4.8}$$

4.2 Other Modules

4.2.1 IF to Baseband

As the FPGA received the IF samples from the ADC, the downconversion to complex baseband was achieved using the sign reversal pattern discussed in Section 2.1. The pattern, $[1, j, -1, -j]$, was implemented with a counter that counted from 0 to 3. When the count was 0, the ADC sample was output over the I channel; when the count was 1, the ADC sample was output over the Q

channel, and so forth.

4.2.2 Matched Filter

The filter used on the transmit side was the 513-coefficient SRRC pulse described previously. However, in order to conserve FPGA resources, the receiver pulse was truncated to 257 coefficients by removing the first and last 128 coefficients. Because the outer coefficients' amplitudes were much smaller than the center coefficients, the effect of this pulse truncation on system performance was minimal (although future work may reveal that this extra windowing needs to be removed due to frequency side lobes). The remaining coefficients were scaled by the inverse of the CORDIC scale factor Z described in section 4.1.1. Embedding this scale factor in the filter coefficients eliminated the need for the CORDIC algorithm to remove the scaling, which reduced latency and resource utilization.

4.2.3 ASM Detector

The bit sequence ASM detector discussed in Section 3.1.4 was employed in this system. Using the lock points from Figure 3.5, rotations of 30 degree intervals were applied to the ASM. This resulted in 12 256-bit sequences which were stored as constants on the FPGA. Because the length of the ASM was 256 bits, a register of the 64 most recent decisions was kept for the ASM detector.

The comparisons of the decision register and the 12 possible versions of the ASM were performed using a bitwise XOR operation. The XOR output is high at each bit where the two inputs do not match. Therefore, the next step was to count the number of 1's in each bitwise XOR output to find if the detection threshold was met for any ASM rotation.

The counting of 1's in each 256-bit XOR output was broken down into 8 (i.e. $\log_2(256)$) accumulation stages. After the number of differences between the decision register and each ASM rotation was counted, the minimum number of differences was found. If this minimum was less than the detection threshold (which was set as 64), then the detection was considered positive, and

the offset of the corresponding ASM rotation was corrected in the PLL. Otherwise, the detection was considered negative.

In addition to phase ambiguity resolution, the ASM detector is used to mark the beginning of a new frame since each frame begins with the ASM. This is used by the output control module (Section 4.2.5) to keep the ASM symbols from being fed to the LDPC decoder.

4.2.4 LLR Calculation

The LLR approximation shown in (2.6) was modified for FPGA implementation. The goal was reduced latency; this was one of the final modules implemented, and a report of FPGA utilization up to this point revealed that there was a large enough number of DSP slices remaining to prioritize a smaller output delay. This reduced the amount of bookkeeping required for the output control module.

The implemented LLR calculation is shown in (4.9), where $\mathbf{a}^1$ and $\mathbf{a}^0$ are the sets of 16-APSK symbols having a 1 and 0 in the ith bit position respectively.

$$\lambda_i = \max\left\{2\,\mathrm{Re}[\langle r, \mathbf{a}^1\rangle] - ||\mathbf{a}^1||^2\right\} - \max\left\{2\,\mathrm{Re}[\langle r, \mathbf{a}^0\rangle] - ||\mathbf{a}^0||^2\right\} \tag{4.9}$$

One clear difference between (2.6) and (4.9) is that (4.9) does not require knowledge of the noise. Ignoring the σ^2 term in (2.6) is permissible because the scaled-min algorithm of the LDPC decoder is insensitive to a scaling of the input LLRs. However, the main difference between (2.6) and (4.9) is that (2.6) requires finding $\mathbf{a}^1_{min}$ and $\mathbf{a}^0_{min}$, i.e. the nearest constellation points with a 1 and 0 in each bit position, before carrying out the LLR calculation. Instead, (4.9) computes the 1 and 0 components of the LLR for each set of symbols $\mathbf{a}^1$ and $\mathbf{a}^0$ and selects the maximum 1 and 0 terms for LLR calculation. The computational cost of (4.9) was reduced by storing $\mathbf{a}^1$ and $\mathbf{a}^0$ as constants.

4.2.5 Output Control

The goal of the synchronization system is to send the LDPC decoder LLRs from the received LDPC packets. The LLR calculation module is triggered with every new decision and has no knowledge of a successful ASM detection. This means that LLRs for the ASM symbols are computed. The LDPC decoder does not need to decode the ASM because it does not contain any information; in fact, the ASM must not be included in the decoder input due to the expected block length. Therefore, a buffer is needed between the LLR calculations and the LDPC decoder input to ensure that only the LDPC codewords are fed into the decoder.

This buffer was achieved by writing every new LLR to a block of memory and counting each time a new LLR arrived. The values were stored until the ASM detector went high. When the ASM detector went high, the saved LLR values were read out excluding the most recent 256 LLRs. This is because if the ASM detector goes high, then the previous 64 decisions (i.e. 256 LLRs) were all ASM symbols.

4.3 Clock and Reset

In a synchronous HDL design, each component is triggered at a regular rate by the edge of a clock signal. Therefore, the rate of the clock is an extremely significant design parameter. System timing requirements and hardware capabilities are the main factors that determine the clock rate.

The buffers of the black box PCIe interface of this system are clocked at 125 MHz. Given the system downsampling factor $N/2 = 16$ and the constraint that the interpolator update $\mu(k+1)$ must be computed before the next interpolation, the maximum time for the interpolator update computation is 16 samples$/$125 Msamples/sec $=$ 128 ns. If a common 125 MHz clock were used between the PCIe buffers and the synchronization system, the 16 clock cycles would not be enough for the synchronization system to complete its loop in time. Therefore, this system was implemented as a multirate design with a higher clock frequency for some components. Specifically, the modules

of the symbol timing synchronization system, including the interpolator, the decision block, and the timing PLL, were clocked at 400 MHz. This clock rate was found to meet performance timing requirements while also satisfying the FPGA hardware timing constraints. The phase recovery system's PLL required more clock cycles than the timing recovery PLL due to the modulo-2π operation and the addition of the ASM. These additional operations were enough to make the phase synchronization system unable to complete a loop in 128 ns. Consequently, the carrier phase synchronization system was pipelined across symbol times as previously mentioned because it tolerated a delay in phase error estimation without a noticable impact on performance.

Operating at the slower clock rate simplifies implementation because a slower clock rate eases the hardware timing constraints. Therefore, modules without strict system timing requirements, i.e. the ASM detector and LLR computation block, were clocked at the slower clock rate. The CORDIC rotation block was also clocked at the slower clock rate because the CORDIC hardware was limited to a frequency less than 400 MHz.

Figure 4.3 shows the connections between each module. The main purpose of Figure 4.3 is to depict the clock domains of the design. A multirate design must handle signals crossing between clock domains. Signals crossing from the slower clock domain to the faster clock domain present different issues than signals crossing from the faster to slower clock domain.

For a signal crossing from the slower to faster domain, the concern is that the signal could still be in a transient state at the faster clock's edge. This could result in a metastable state at the faster domain's register. Slower-to-faster crossings are handled in this design with the double flip-flop technique shown in Figure 4.2. If the first flip-flop in the faster clock domain is metastable, the second flip-flop ensures the signal is stabilized before being fanned out to the rest of the design.

For a signal crossing from the faster to the slower clock domain, the concern is that signal transitions in between slower clock edges would not be seen in the slower clock domain. This is addressed by passing the faster domain signal through enough faster clocked flip-flops so that each

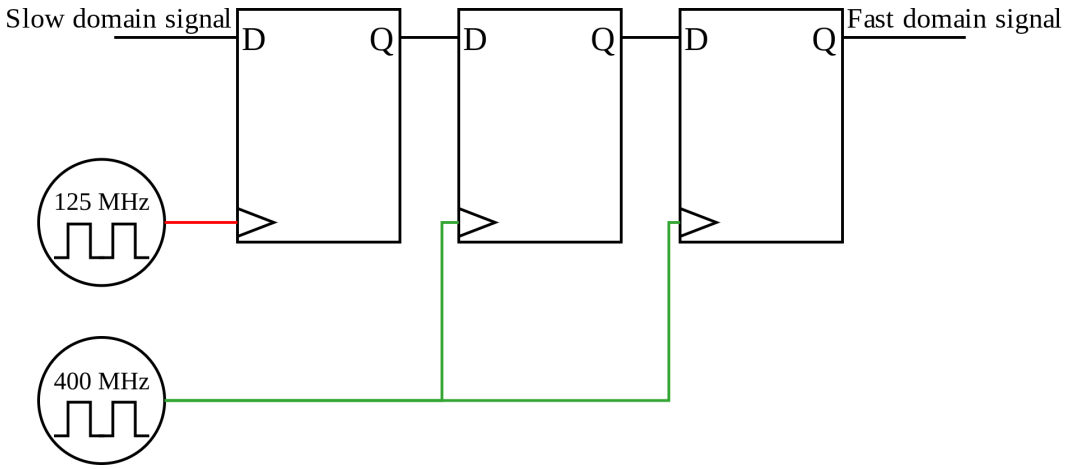

Figure 4.2: Double Flip-Flop for Crossing from Slower to Faster Clock Domain

faster domain transition is held through a slower clock edge. The number of required registers is determined by the relationship between the two clocks. In this design, because the faster clock is 3.2 times faster than the slower clock, at least 4 registers are required to cross from the faster clock domain to the slower clock domain.

Although not shown in Figure 4.3 to reduce the number of wires, a global reset signal is used for every module in the design. The reset signal clears all registers when a transmission ends, as well as initializing registers at power-up. The global reset is set within the black box PCIe interface.

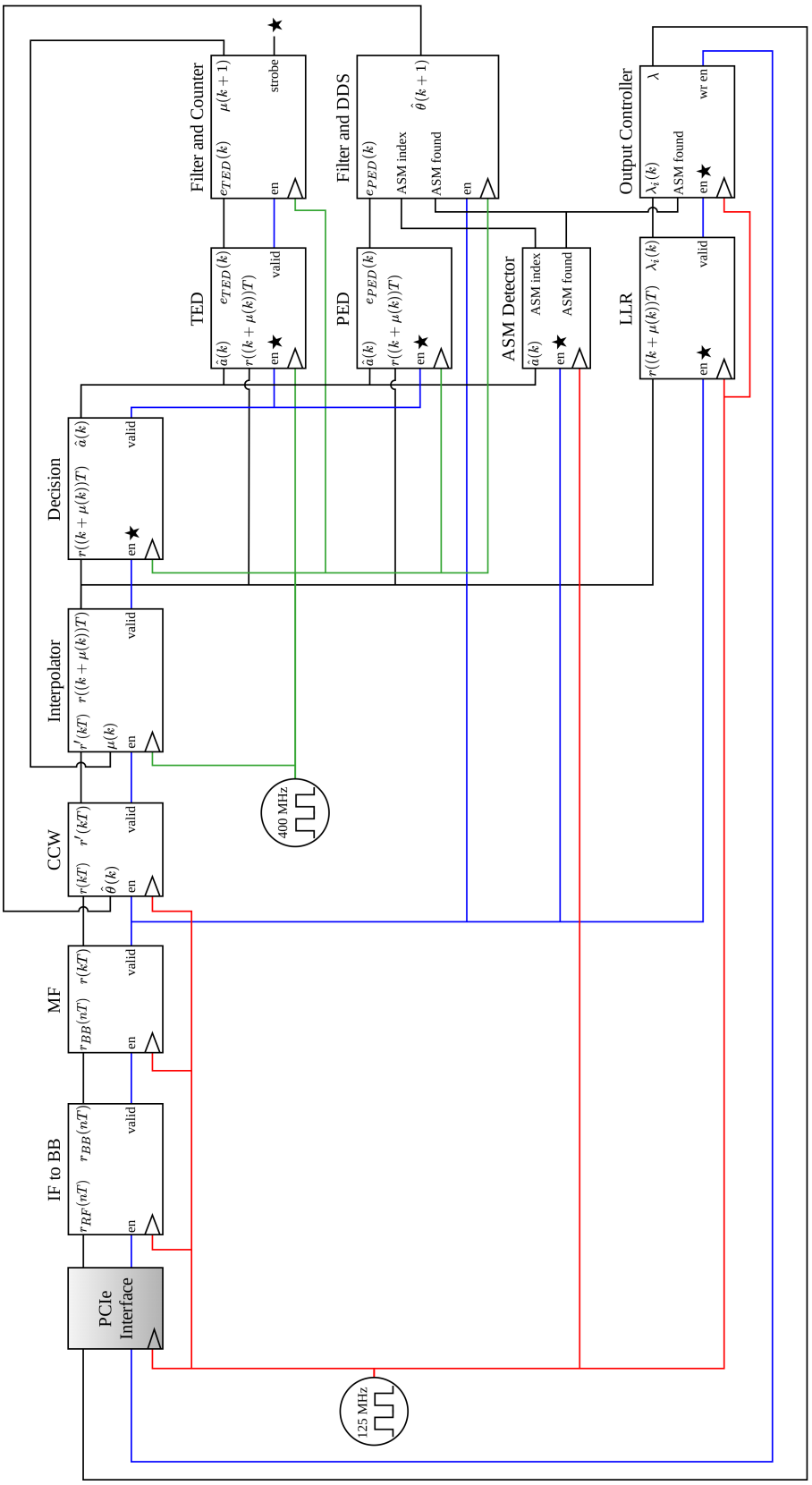

Figure 4.3: FPGA Implementation Block Diagram

4.4 FPGA Resource Utilization

FPGA performance can be limited by the design if the utilization of certain resources approaches 100%. If the design becomes too congested, the synthesis tool will be unable to route signals in a way that meets timing constraints. FPGA resource utilization has been considered and mentioned throughout this chapter; Table 4.3 summarizes the utilization of this design in terms of look-up tables (LUT), flip-flops (FF), block memory (BRAM), and DSP slices as reported by the implementation tool.

Table 4.3: FPGA Resource Utilization

Resource	Utilization (%)
LUT	9.50
FF	6.42
BRAM	5.34
DSP Slice	37.57

Chapter 5

Test Setup

This chapter describes the testing environment for the synchronization system.

5.1 Test Data

5.1.1 LDPC Parameters

Two considerations for an LDPC system are the code rate and the block length. The code rate is the ratio of information bits to total (i.e. information plus parity) bits per block. The block length is the length of each LDPC codeword. A larger block size results in better performance at the expense of increased decoding complexity. The LDPC system, defined in [13], uses a block length of 4096 information bits and a code rate of 4/5. These parameters were chosen based on the balance between a high code rate and the power backoff capabilities shown in [6].

5.1.2 Payload

The system's LDPC block length is 4096 information bits. Therefore, an appropriate method for random bit generation is a 9-bit pseudo-random (PN9) sequence which results in $2^9 - 1 = 511$ unique 8-bit words. Repeating one of these words results in 512 8-bit words, or 1024 16-APSK symbols. A PN9 sequence is described by the polynomial $x^9 + x^5 + 1$. With a code rate of 4/5, the resulting frame length is $4096/(4/5) = 5120$ bits after LDPC encoding.

5.1.3 Attached Synchronization Marker

To mark the beginning of a new LDPC frame and assist in phase ambiguity resolution, an attached synchronization marker (ASM) is added to the beginning of each frame. The ASM is 256 bits long, and it is specified in [8]. The purpose of the ASM is purely functional: It serves as a flag for a new frame in the receiver, and it is a known reference for phase ambiguity resolution, but it does not contain any information and therefore is not LDPC encoded or decoded.

With the addition of the ASM, the final ratio of information bits to total bits is $R_b = 4096/(5120+256) = 16/21$.

5.1.4 Data Rate

The minimum data rate was given as 5 megabits per second (Mbps). To simplify system design, the implemented data rate was based on an integer upsampling factor of 32 samples per symbol. Using the previously derived sampling rate of 93.333 MHz, the resulting data rate was 8.888 Mbps, as shown below:

$$F_s = 93.333 \text{ MHz}$$

$$N = 32 \text{ samples/symbol}$$

$$F_s/N = 2.917 \text{ Msymb/sec}$$

$$4F_s/N = 11.666 \text{ Mbps (total)}$$

$$R_b(11.666 \text{ Mbps}) = 8.888 \text{ Mbps (information)}$$

5.2 Equipment Setup

In order to isolate the receiver, a signal generator was used to output the test signal at IF into the ADC. The signal generator was set to an output power level of -20 dBm. Gaussian noise with a given variance could be added at the signal generator for BER performance tests. The ADC output

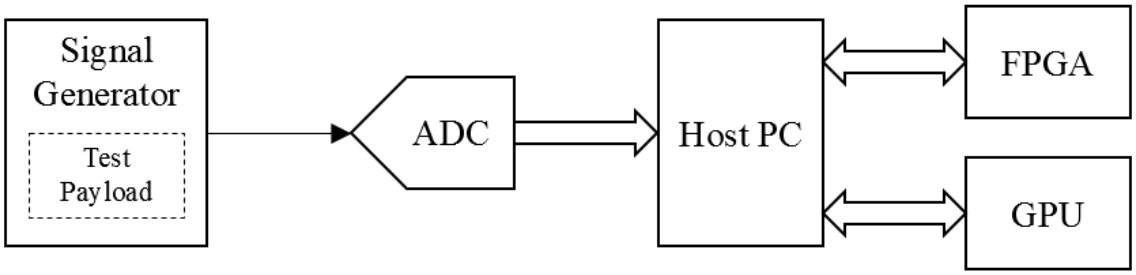

Figure 5.1: Test Equipment Setup

was transferred to a host PC, where the digitized data was normalized to unit energy.

After the ADC, the data in the receiver passes through three main blocks: a host PC, the FPGA, and a graphics processing unit (GPU). The host serves as a mediator between the other system blocks, and data is passed between blocks via peripheral component interconnect express (PCIe) bus. The host collects the samples from the ADC and sends them to the FPGA for processing. After demodulation, the FPGA sends its computed LLRs to the host, and the host sends them to the GPU. The GPU is where the LDPC decoding is performed. The GPU sends the decoded decisions to the host for analysis, i.e. BER measurement.

A block diagram of the test setup is shown in Figure 5.1, where block arrows represent PCIe bus communication.

5.3 Simulation Description

Floating point simulation was used as a benchmark for FPGA synchronization performance. In order to directly compare the simulation and FPGA implementation while isolating the synchronization system, the FPGA downsampled matched filter outputs were captured and input to the simulated synchronization system. As opposed to beginning the simulation with the entire set of samples acquired by the ADC, this approach ensured that only synchronization system performance was observed without additional quantization noise from the matched filter.

Chapter 6

Results and Analysis

This chapter presents and discusses the results of the FPGA implementation of the synchronization system. Synchronization system performance is observed here relative to simulation in terms of the demodulated signal's BER, error vector magnitude (EVM), and the settling time and steady-state values of the interpolation fractional interval $\mu(k)$ and the phase correction estimate $\hat{\theta}(k)$. EVM is a measure of the average distance from a demodulated vector to its nearest constellation point for all received symbols. EVM is 0 if the received point corresponds perfectly to a constellation point. The EVM is normalized by the maximum constellation amplitude and expressed as a percentage as shown in (6.1), where K is the number of symbols in the received signal.

$$\text{EVM} = \left(\frac{1}{K} \sum_{k=0}^{K-1} \frac{\sqrt{|r(kT_s) - \hat{a}(k)|^2}}{\max_{a \in \mathbb{A}} |a|} \right) * 100\% \tag{6.1}$$

Figure 6.1 and Figure 6.2 show the simulated and FPGA implemented demodulated constellations respectively. These constellations show the data before and after the synchronization system locks, where the locking point was based on the settling time of $\hat{\theta}(k)$ and $\mu(k)$. Figure 6.3 shows the phase correction estimate $\hat{\theta}(k)$ from simulation and implementation. Only the acquisition stage is shown. Similarly, Figure 6.4 shows the acquisition of the interpolator fractional interval $\mu(k)$. Figure 6.5 shows the BER performance of the implementation compared to simulations from [6].

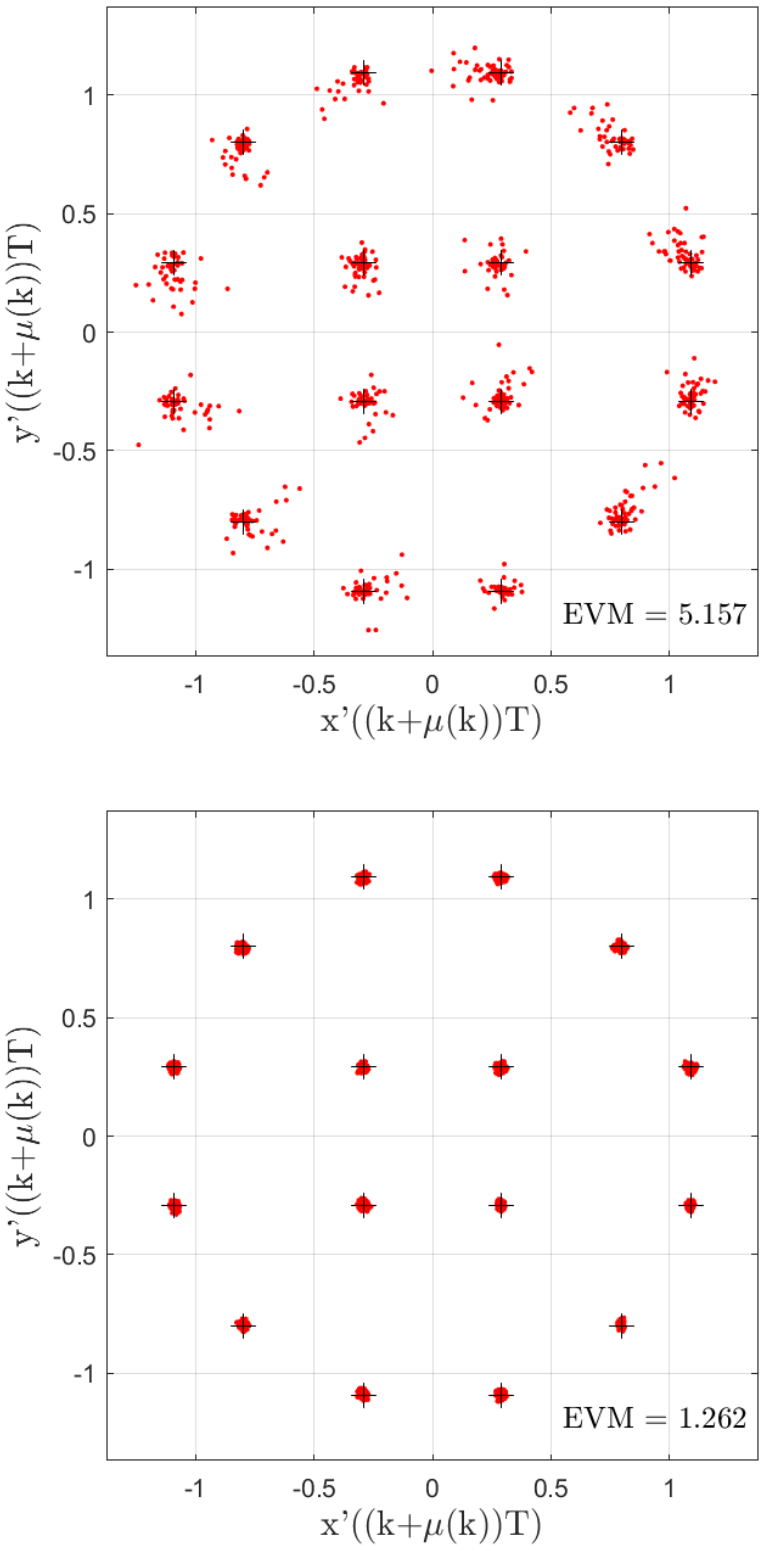

Figure 6.1: Demodulated Vectors, Acquiring (top) and Locked (bottom), Simulation

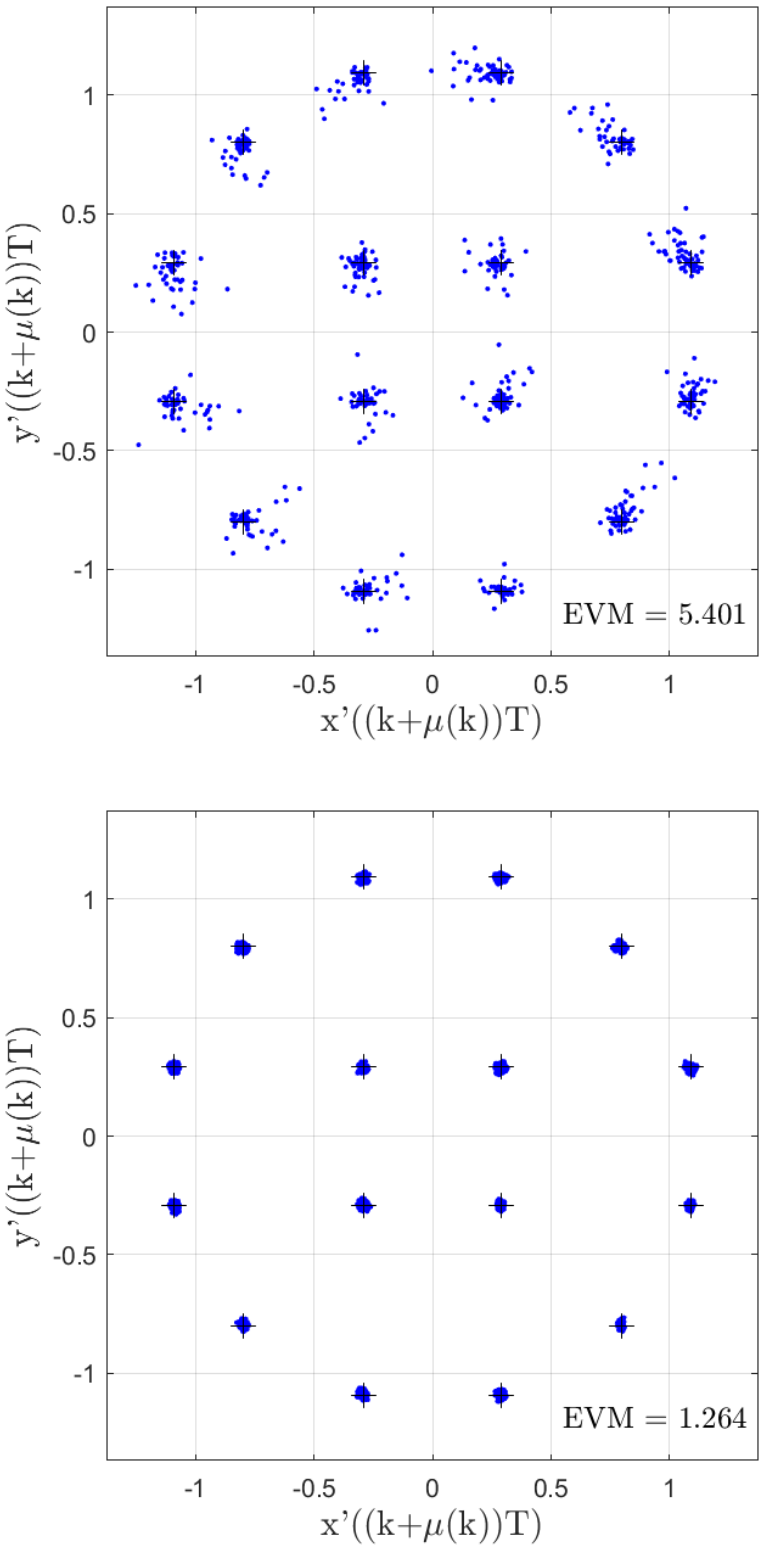

Figure 6.2: Demodulated Vectors, Acquiring (top) and Locked (bottom), Implementation

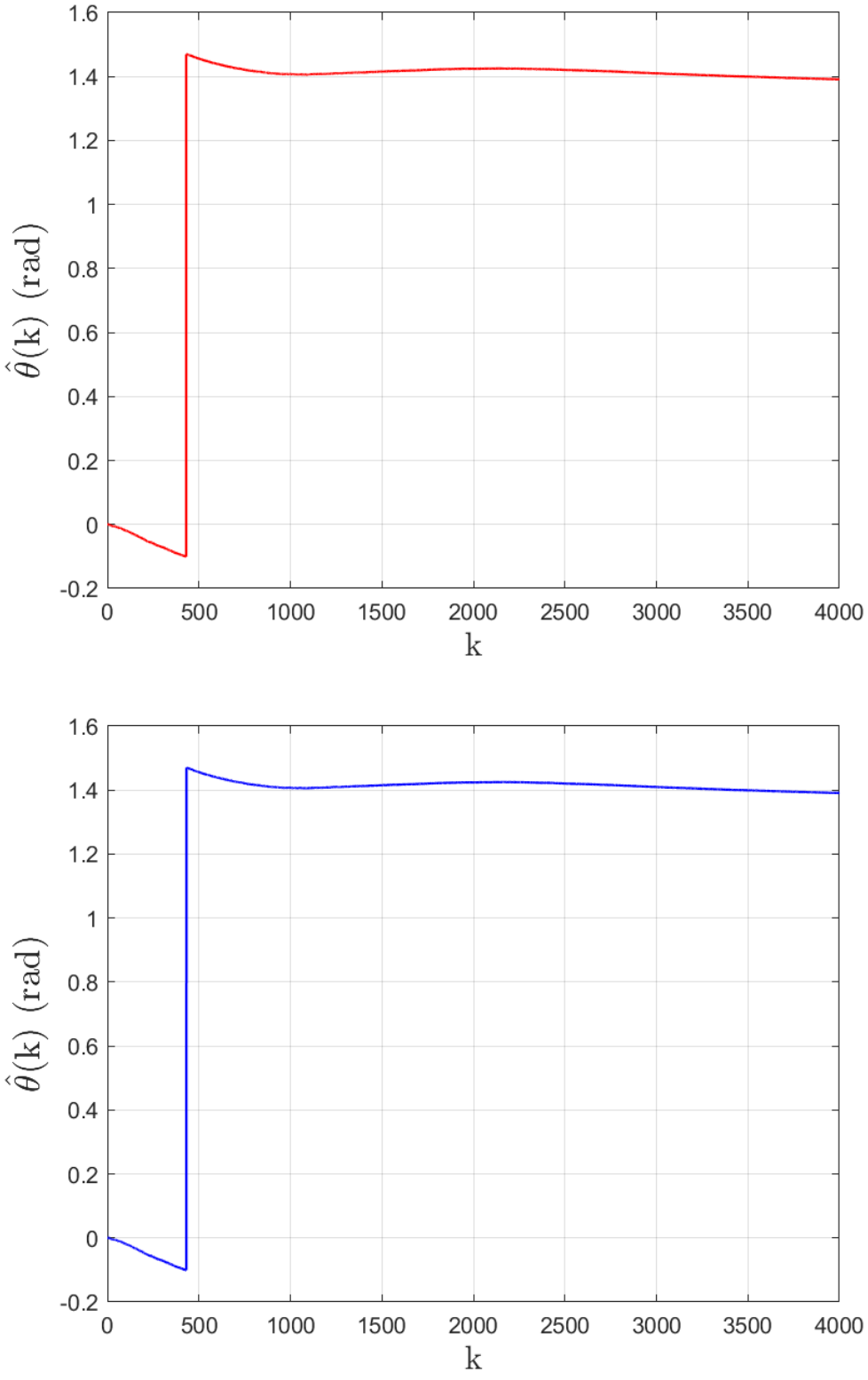

Figure 6.3: Phase Correction Estimate, Simulation (top) and FPGA Implementation (bottom)

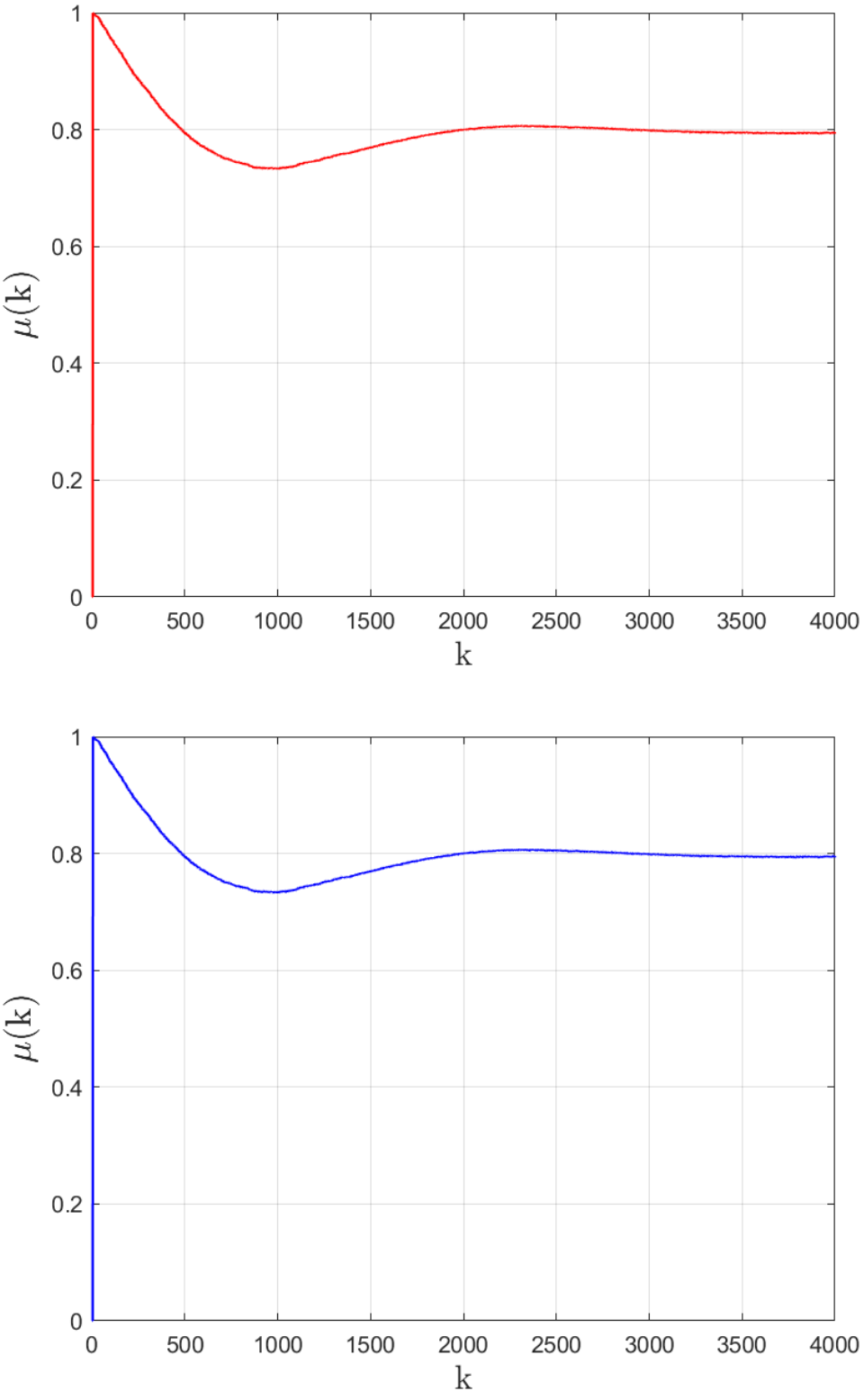

Figure 6.4: Interpolator Fractional Interval, Simulation (top) and FPGA Implementation (bottom)

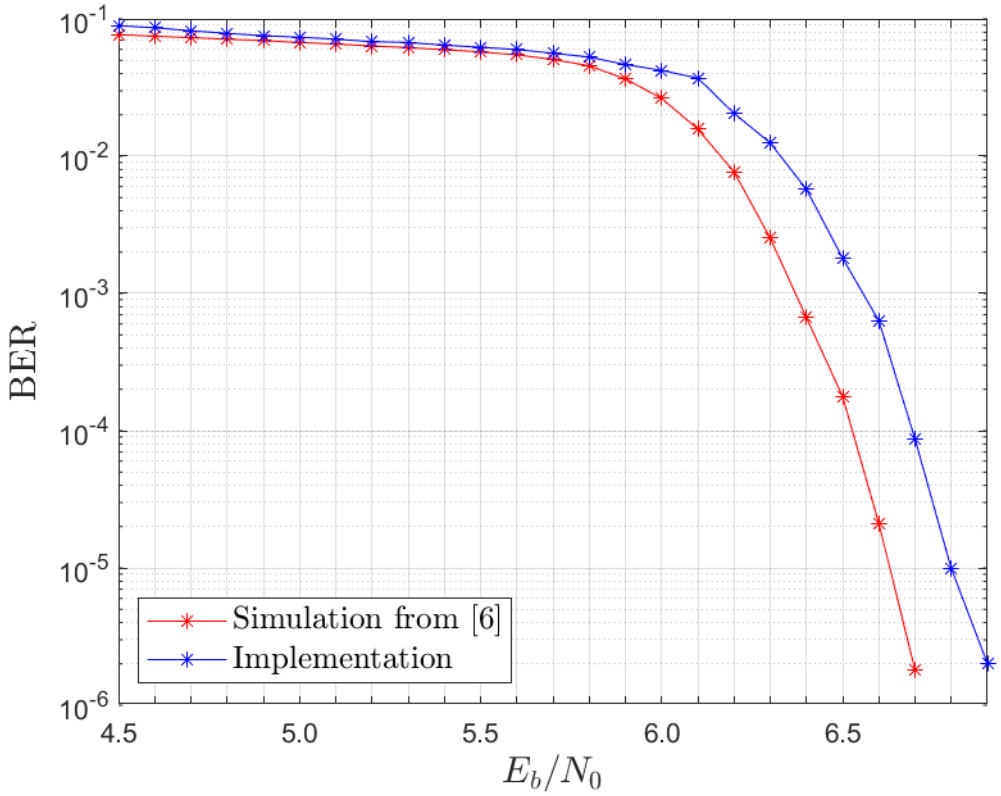

Figure 6.5: BER Performance

The constellations of Figures 6.1 and 6.2 show that the EVM was significantly reduced after the synchronization system locked. During acquisition, points can be seen close to decision region boundaries. This indicates potential errors, especially in a noisy system. More important in terms of bit error performance is the large change in the phase correction estimate shown in Figure 6.3. This indicates the point where the ASM was detected and the corresponding ASM phase adjustment was applied. The figure shows an ASM phase adjustment of $\pi/2$ radians. This means that every point before the ASM detection was offset from its transmitted constellation point by $\pi/2$, resulting in a symbol error rate of 100%. This emphasizes the discussion of phase ambiguity from Section 3.1.4: Although the demodulated vectors form a constellation that appears to indicate correct demodulation, there is in reality a constant offset at every point.

The results show that the FPGA implementation performed nearly identically to the simulation. The FPGA-implemented timing and carrier phase PLLs exhibited the same acquisition time (both

within one LDPC frame time) and steady-state values as the simulation, and the EVM is within 0.002% after lock. The larger difference in EVM during acquisition is most likely from quantization within the PLLs. In early runs, the PLL accumulators still have small enough magnitude to cause a discrepancy between floating point and 16-bit fixed point numbers. The FPGA BER curve is approximately 0.2 dB worse than the simulated curve. This is most likely because the work in [6] (from which the simulated curve was obtained) was performed with end-to-end floating point numbers and perfect synchronization. The implemented BER curve is subject to quantization from the ADC, FPGA, and GPU.

Chapter 7

Conclusion and Future Work

This thesis has covered the FPGA implementation of a carrier phase and symbol timing synchronization subsystem in a 16-APSK communications receiver. The work was part of an investigation of the benefits of LDPC coded APSK in terms of spectral efficiency for a C-band aeronautical telemetry system. The motivation for a synchronization system was presented, its theory of operation was discussed, and the FPGA implementation of each synchronization block was explained. Finally, the FPGA implementation was shown to successfully demodulate a test signal acquired from an ADC. Performance was shown to be nearly identical to floating point simulation.

Future work will seek to increase the maximum correctable frequency offset of the synchronization system presented in this thesis. The PLL-based method of this work has been shown to correct a frequency offset of up to 30 kHz [14]. A practical system could experience a Doppler shift larger than 30 kHz. The work in [15] employs an FFT-based approach to provide an initial frequency offset correction. Simulations have shown that when this is combined with the PLL-based method of this thesis, the system can account for a larger frequency offset. The FPGA resource utilization report of Table 4.3 indicates that the FPGA could easily accommodate additional modules on top of the design presented in this thesis.

Bibliography

[1] M. Rice, C. Hogstrom, and C. Nash, "On IF-to-Baseband Translation and Resampling in Sampled-Data Receivers," in *Proc. Int. Telemetering Conf.*, (Las Vegas, NV), Oct. 2017.

[2] M. Rice, *Digital Communications: A Discrete Time Approach*. Pearson, 2009.

[3] C. Josephson, "On the Design of a Square-Root Nyquist Pulse Shaping Filter for Aeronautical Telemetry," in *Proc. Int. Telemetering Conf.*, (Las Vegas, NV), Oct. 2017.

[4] J. Hamkins, "Performance of Low-Density Parity-Check Coded Modulation," in *2010 IEEE Aerospace Conference*, pp. 1–14, 2010.

[5] T. K. Moon, *Error Correction Coding: Mathematical Methods and Algorithms*. John Wiley & Sons, 2005.

[6] S. Pathak, "A Performance and Channel Spacing Analysis for LDPC Coded-APSK," Master's thesis, University of Kansas, 2018.

[7] European Telecommunications Standards Institute (ETSI), *Digital Video Broadcasting (DVB), Second generation framing structure, channel coding and modulation systems for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2)*, 2006.

[8] Consultive Committee for Space Data Systems (CCSDS), *TM Synchronization and Channel Coding*. 131.0-B-1, Sep. 2003.

[9] K. Andrews, V. Stanton, S. Dolinar, V. Chen, J. Berner, and F. Pollara, "Turbo-Decoder Implementation for the Deep Space Network," in *IPN Progress Report 42-148*, JPL, 2002.

[10] F. Gardner, *Demodulator Reference Recovery Techniques Suited for Digital Implementation*. European Space Agency, 1988. ESTEC Contract No. 6847/86/NL/DG.

[11] C. W. Farrow, "A Continuously Variable Digital Delay Element," in *IEEE International Symposium on Circuits and Systems*, pp. 2641–2645 vol.3, 1988.

[12] J. Volder, "The CORDIC Trigonometric Computing Technique," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 3, pp. 330–334, 1959.

[13] Consultive Committee for Space Data Systems (CCSDS), *Low Density Parity Check Codes for Use in Near-Earth and Deep Space Applications*. 131.1-O-2, Sep. 2007.

[14] M. Rice, B. Redd, and X. Briceño, "On Carrier Frequency and Phase Synchronization for Coded 16-APSK in Aeronautical Mobile Telemetry," in *Proc. Int. Telemetering Conf.*, (Las Vegas, NV), Oct. 2019.

[15] B. Redd, J. Ebert, and A. Twitchell, "DFT-Based Frequency Offset Estimators for 16-APSK," in *Proc. Int. Telemetering Conf.*, (Las Vegas, NV), Oct. 2019.